

Bit-Parallel Vector Composability for Neural Acceleration

Soroush Ghodrati Hardik Sharma[†] Cliff Young[‡] Nam Sung Kim[§] Hadi Esmaeilzadeh

Alternative Computing Technologies (ACT) Lab

University of California, San Diego [†]Bigstream, Inc. [‡]Google Inc. [§]University of Illinois Urbana-Champaign

soghodra@eng.ucsd.edu hardik@bigstream.co cliffy@google.com nskim@illinois.edu hadi@eng.ucsd.edu

Abstract—Conventional neural accelerators rely on isolated self-sufficient functional units that perform an atomic operation while communicating the results through an operand delivery-aggregation logic. Each single unit processes all the bits of their operands atomically and produce all the bits of the results in isolation. This paper explores a different design style, where each unit is only responsible for a slice of the bit-level operations to interleave and combine the benefits of bit-level parallelism with the abundant data-level parallelism in deep neural networks. A dynamic collection of these units cooperate at runtime to generate bits of the results, collectively. Such cooperation requires extracting new grouping between the bits, which is only possible if the operands and operations are vectorizable. The abundance of Data-Level Parallelism and mostly repeated execution patterns, provides a unique opportunity to define and leverage this new dimension of Bit-Parallel Vector Composability. This design intersperses bit parallelism within data-level parallelism and dynamically interweaves the two together. As such, the building block of our neural accelerator is a Composable Vector Unit that is a collection of Narrower-Bitwidth Vector Engines, which are dynamically composed or decomposed at the bit granularity. Using six diverse CNN and LSTM deep networks, we evaluate this design style across four design points: with and without algorithmic bitwidth heterogeneity and with and without availability of a high-bandwidth off-chip memory. Across these four design points, Bit-Parallel Vector Composability brings (1.4× to 3.5×) speedup and (1.1× to 2.7×) energy reduction. We also comprehensively compare our design style to the Nvidia’s RTX 2080 TI GPU, which also supports INT-4 execution. The benefits range between 28.0× and 33.7× improvement in Performance-per-Watt.

Index Terms—Acceleration, neural networks, bit-flexibility

I. INTRODUCTION

The growing body of neural accelerators [1–3, 6, 7, 9–11, 14, 16, 19] exploit various forms of Data-Level Parallelism (DLP) that are abundant in Deep Neural Networks. For instance, Google’s TPU [9] extracts data-level parallelism across the lanes of its systolic array, Microsoft’s Brainwave [6] builds a dataflow architecture from vectorized units, and GPUs have been long designed for Single-Instruction Multiple-Data (SIMD) execution model. Nonetheless, these various organization still rely on isolated self-sufficient units that process all the bits of input operands and generate all the bits of the results. These values, packed as atomic words, are then communicated through an operand delivery-aggregation interconnect for further computation. This paper sets out to explore a different design where each unit in vectorized engines is only responsible for processing a bit-slice. This design offers an opening to explore the interleaving of Bit-Level Parallelism with DLP for neural acceleration. Such an interleaving opens a new tradeoff space where the complexity of Narrow-Bitwidth Functional Units needs to be balanced with respect to the overhead of bit-level aggregation as well as the width of vectorization.

Additionally, the proposed approach creates a new opportunity for exploring bit-flexibility in the context of vectorized execution.

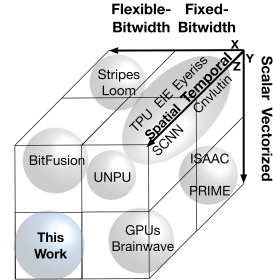


Fig. 1: The landscape of DNN accelerators and how this work fits in the picture.

As Figure 1 illustrates, the landscape of neural accelerators can be depicted in the three dimensions of Type of Functional Units (Scalar [1, 2, 7, 9, 14] v.s. Vectorized [3, 6, 16]), Bit Flexibility¹ (Fixed Bitwidth v.s. Flexible Bitwidth), and Composability (Temporal [10, 11, 17] vs. Spatial [19]). This work aims to fill the vacancy for Vectorized Bit-Flexible Spatial Composability. In fact, the interleaving of the bit-level parallelism with DLP creates the new opportunity to explore bit-level composability where a group of bit-parallel vector units *dynamically* form collaborative groups to carry out SIMD operations. Recent inspiring research investigates bit-level composability for a scalar operation (a single MAC unit) both in spatial and temporal mode. However, the space where composability (bit-level parallelism) meets vectorization (data-level parallelism) is not explored. Our design style opens this space by breaking up the vector units to narrower bitwidth engines. Since the building block is a narrow bitwidth vector engine, another opportunity arises to support execution well below eight bits. That is leveraging the algorithmic insight that heterogenous assignment of bitwidths below eight to DNN layers can reduce their computational complexity while preserving accuracy [4, 5, 8, 12, 15]. Fixed bitwidth designs cannot tap into this level of efficiency where each unit is only processing the fewest number of bits that can preserve the accuracy.

We evaluate the proposed concepts with and without availability of algorithmic bitwidth heterogeneity. Experimentation with six real-world DNNs shows that bit-parallel vector composability provides 40% speedup and energy reduction compared to design with the same architecture (systolic) without support for the proposed composability. When bit flexibility is applicable because of the heterogenous bitwidths in the DNN layers, our design provides 50% speedup and 10% energy reduction compared to BitFusion [19], the state-of-the-art architecture that supports scalar bit-level composability in systolic designs².

¹Bit flexibility can be defined as the capability of a functional unit in reconfiguring itself *at runtime* to support different bitwidths.

²We do not compare to BitFusion when algorithmic bitwidth heterogeneity is not available, BitFusion would underperform compared to a non-composable design.

When a high bandwidth off-chip memory is utilized, the baseline design only enjoys 10% speedup and 30% energy reduction, respectively. However, bit-parallel vector-composability better utilizes the boosted bandwidth and provides $2.1\times$ speedup and $2.3\times$ energy reduction. With algorithmic bitwidth heterogeneity our design style provides $2.4\times$ speedup and 20% energy reduction, compared to BitFusion, while it also utilizes the same high bandwidth memory. Finally, we compare different permutations of our design style with respect to homogenous/heterogenous bitwidth and off-chip memory bandwidth to the Nvidia's RTX 2080 TI GPU which also supports INT-4 execution. Benefits range between $28.0\times$ and $33.7\times$ higher Performance-per-Watt for four possible design points.

II. BIT-PARALLEL VECTOR COMPOSABILITY

This paper builds upon the fundamental property of *vector dot-product operation* – the most common operation in DNNs – that vector dot-product with wide-bitwidth data types can be decomposed and reformulated as a summation of several dot-products with narrow-bitwidth data types. The element-wise multiplication in vector dot-product can be performed independently, exposing *data-level parallelism*. This work explores another degree of parallelism – *bit-level parallelism* (BLP) – wherein individual multiplications can be broken down at *bit-level* and written as a summation of narrower bitwidth multiplications. Leveraging this insight, this work studies the interleaving of both data-level parallelism in vector dot-product with bit-level parallelism in individual multiplications to introduce the notion of *bit-parallel vector composability*. This design style can also be exploited to support *runtime-flexible* bitwidths on the underlying hardware. The rest of this section details the mathematical formulation for bit-parallel vector composability. **Fixed-Bitwidth bit-parallel vector composability.** Digital values can be expressed as the summation of individual bits multiplied by powers of two. Hence, a vector dot-product operation between two vectors, $\vec{X}$, $\vec{W}$ can be expressed as follows:

$$\vec{X} \bullet \vec{W} = \sum_i (x_i \times w_i) = \sum_i ((\sum_{j=0}^{bw_x-1} 2^j \times x_i[j]) \times (\sum_{k=0}^{bw_w-1} 2^k \times w_i[k])) \quad (1)$$

Variables bw_x and bw_w represent the bitwidths of the elements in $\vec{X}$ and $\vec{W}$, respectively. Expanding the bitwise multiplications between the elements of the two vectors yields:

$$\vec{X} \bullet \vec{W} = \sum_i (\sum_{j=0}^{bw_x-1} \sum_{k=0}^{bw_w-1} 2^{j+k} \times x_i[j] \times w_i[k]) \quad (2)$$

Conventional architectures rely on compute units that operate on all bits of individual operands, and require complex left-shift operations followed by wide-bitwidth additions as shown with an underline in Equation 2. By leveraging the associativity property of the multiplication and addition, we can cluster the bit-wise operations that share the same significance position together and factor out the power of two multiplications. In other words, this clustering can be realized by swapping the order of $\sum_i$ and $\sum_j \sum_k$ operators.

$$\vec{X} \bullet \vec{W} = \sum_{j=0}^{bw_x-1} \sum_{k=0}^{bw_w-1} 2^{j+k} \times (\sum_i x_i[j] \times w_i[k]) \quad (3)$$

Leveraging this insight enables the use of significantly less complex, narrow-bitwidth, compute units (1-bit in the equation), exploiting bit-level parallelism, amortizing the cost of left-shift and wide-bitwidth addition. Breaking down the dot-product is not limited

to single bit and elements of the vectors can be *bit-sliced* with different sizes. As such, Equation 3 can be further generalized as:

$$= \sum_{j=0}^{\frac{bw_x}{\alpha}-1} \sum_{k=0}^{\frac{bw_w}{\beta}-1} 2^{\alpha j + \beta k} \times (\sum_i x_i[\alpha j : (\alpha+1)j] \times w_i[\beta k : (\beta+1)k]) \quad (4)$$

Here, α and β are the bit-slices for operands x_i and w_i for the narrow-bitwidth compute units, respectively. Figure 2-(a) graphically illustrates bit-parallel vector composability using an example of a vector dot-product operation ($\vec{X} \bullet \vec{W} = \sum_i x_i \times w_i$) between two vectors of $\vec{X}$ and $\vec{W}$, each of which constitutes two 4-bit elements. As it is shown with different shades, each element can be bit-sliced and broken down into two 2-bit slices. With this bit-slicing scheme, the original element (x_i or w_i) can be written as $2^2 \times bsl_{MSB} + 2^0 \times bsl_{LSB}$, where $bsls$ are the bit-slices and they are multiplied by powers of two based on their significance position. With the aforementioned bit-slicing scheme, performing a product operation between an element from $\vec{X}$ and another from $\vec{W}$ requires four multiplications between the slices, each of which is also multiplied by the corresponding significance position factor. However, because of the associativity property of the multiply-add, we cluster the bit-sliced multiplications that share the same significance position and apply the power of two multiplicands by shifting the accumulated result of the bit-sliced multiplications.

Flexible-Bitwidth vector composability. The bit-parallel vector composable design style can enable *flexible-bitwidth support at runtime*. Figure 2-(b) shows an example of flexible-bitwidth vector dot-product operation considering *the same number of compute resources (2-bit multipliers, adders, and shifters)* as Figure 2. In Figure 2-(b), a vector dot-product operation between a vector of inputs ($\vec{X}$) that has four 4-bit elements and a vector of weights ($\vec{W}$) that has four 2-bit elements is illustrated. Using the same bit-slicing scheme, the original vector $\vec{X}$ is broken down to two sub-vectors that are required to go under dot-product with $\vec{W}$ and then get shifted and aggregated. However, exploiting 2-bit datatypes for weights compared to 4-bit in the example given in Figure 2-(a), provides $2\times$ boost in compute performance. Bit-parallel vector composability enables maximum utilization of the compute resources, resulting in computing a vector dot-product operation with $2\times$ more elements using the same amount of resources.

The next section discusses the acceleration using this design style.

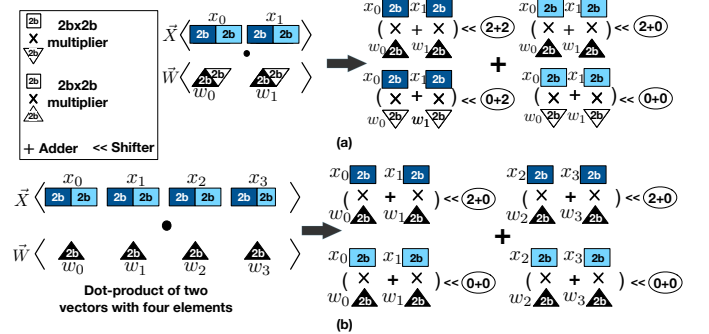
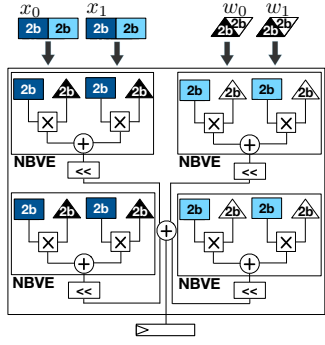
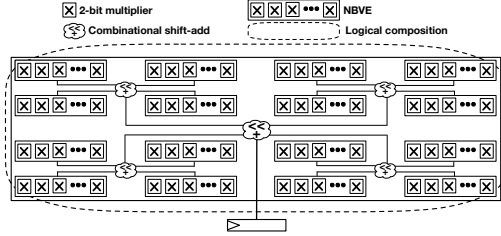


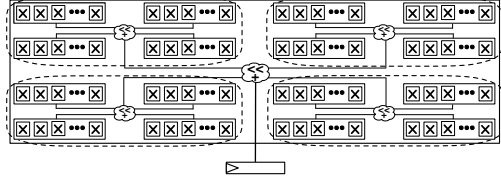
Fig. 2: (a) Fixed-bitwidth bit-parallel vector composability with 2-bit slicing and (b) Bit-Flexible vector composability for 4-b inputs and 2-b weights; 2x improvement in performance compared to fixed 4-bit dot-product.



(a) Hardware realization of CVU for the example in Figure 2-(a)



(b) Homogenous 8-bit mode, conceptual view; All the NBVEs cooperate together and perform an 8-bit $\times$ 8-bit dot product with length of L



(c) Heterogeneous quantized bitwidth mode, conceptual view; four clusters of NBVEs work in parallel and then cooperate together to perform an 8-bit $\times$ 2-bit dot-product with length of $4\times L$

Fig. 3: Composible Vector Unit.

III. ARCHITECTURE

DESIGN FOR BIT-PARALLEL VECTOR COMPOSABILITY

To enable hardware realization for the insight of bit-parallel vector composability, the main building block of our design becomes a Composible Vector Unit (CVU), which performs the vector dot-product operation by splitting it into multiple *narrow-bitwidth* dot-products. As such, the CVU consists of several Narrow-Bitwidth Vector Engines (NBVE) that calculate the dot-product of bit-sliced sub-vectors from the original vectors. The CVU then combines the results from NBVEs according to the bitwidth of each DNN layer. Below, we discuss the micro-architecture.

A. Composible Vector Unit (CVU)

Figure 3-(a) illustrates an instance of the hardware realization of CVU for the vector dot-product example given in Figure 2-(a). In this example, CVU encapsulates 4 Narrow-Bitwidth Vector Engines (NBVE). The number of NBVEs inside a CVU is based on the size of bit-slicing and maximum bitwidth of datatypes (inputs and weights) that have been selected as two and four in this example, respectively. A spatial array of narrow-bitwidth multipliers, connected through an adder-tree, constitute each NBVE. Each narrow-bitwidth multiplier performs a 2-bit $\times$ 2-bit multiplication with a 2-bit-slice of inputs and weights, the result of which then goes to the adder-tree to get aggregated with the outputs of other multipliers inside an NBVE. As such, the NBVE generates a single scalar that is the result of the dot-product operation between two bit-sliced sub-vectors. The CVU then shifts the outputs of NBVEs based on the significance position of their bit-sliced operands and

aggregates the shifted values across all the NBVEs to generate the final result of the vector dot-product operation. In our design, we consider 8-bit as the maximum bitwidth of inputs and weights, commensurate with prior work [9], and 2-bit slicing for the CVU. As such, CVU encapsulates 16 NBVEs that work in parallel. Below, we discuss how the CVU operates when homogenous and heterogeneous bitwidths are exploited.

Homogeneous 8-bit mode of operation. Figure 3(b) illustrates the conceptual view of CVU when 8-bit datatypes are homogeneously exploited for DNN layers. Each NBVE performs a bit-parallel dot-product on 2-bit-sliced sub-vectors of the original 8-bit $\times$ 8-bit dot-product between vectors of length L , generating a scalar. The NBVEs are equipped with shifters to shift their scalar outputs. Finally, to generate the final scalar of the 8-bit dot-product, all the NBVEs in a CVU, globally cooperate together and CVU aggregates their outputs.

Heterogeneous quantized bitwidth mode of operation. When the DNN uses heterogeneous bitwidths (less than 8-bit datatypes) across its layers, the CVU can be dynamically reconfigured to match the bitwidths of the DNN layers at runtime. This includes both the reconfigurations of the shifters and the NBVEs composition scheme. For instance, Figure 3 (c) illustrates a case when 8-bit inputs and 2-bit weights are used, the 16 NBVEs will be clustered as four groups, each of which encapsulates four NBVEs. In this mode, the CVU composes the NBVEs in two levels. At the first level, all the four NBVEs in each cluster are privately composed together by applying the shift-add logic to complete a dot-product operation with a length of L . At the second level, the CVU globally aggregates the outputs of all four clusters and produces the scalar result of dot-product operation between vectors of length $4\times L$. As another example, if 2-bit datatypes are used for both inputs and weights, each NBVE performs an independent dot-product, providing $16\times$ higher performance compared to the homogenous 8-bit mode. To evaluate the benefits of the bit-parallel vector composability and its design trade-offs, we perform a design space exploration for different number of multipliers in an NBVE (L) and choice of bit-slicing and analyze the sensitivity of the CVU's power/area to these parameters, as follows.

B. Design Space Exploration and Tradeoffs

Figure 4 shows the design space exploration for 1-bit and 2-bit slicing, in addition to different lengths of vectors for NBVE from $L=1$ to $L=16$. All the design points in this analysis are synthesized in 500 Mhz and 45 nm node. Y-Axis shows the power and area per one 8-bit $\times$ 8-bit MAC operation performed by CVU normalized to the power/area of a conventional digital 8-bit MAC unit. We sweep the L parameter for 1-bit slicing and 2-bit slicing in the X-axis. Figure 4 also shows the breakdown of power and area across four different hardware logics; multiplication, addition, shifting, and registering. Inspecting this analysis leads us to following key observations:

(1) *Adder-tree consumes the most power/area and might bottleneck the efficiency.* As we observe in both 1-bit and 2-bit slicing, across all the hardware components adder-tree ranks first in power/area consumption. Bit-parallel vector composability imposes two levels of add-tree logic: (a) An adder-tree private to each NBVE that sums the results of narrow-bitwidth multiplications. (b) A global adder-tree that aggregates the outputs of NBVEs to generate the final scalar result. Hence, to gain power/area efficiency, the cost of add-tree requires to be minimized.

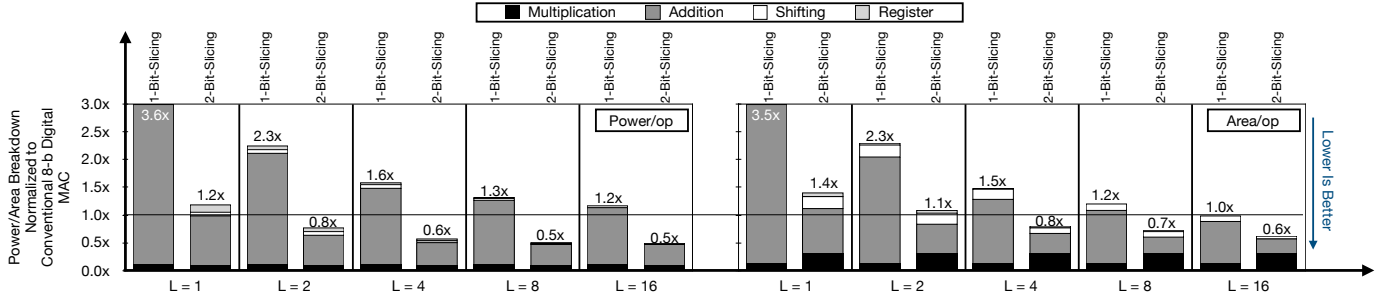


Fig. 4: Design space exploration for size of bit-slicing and vector lengths for vector composability.

(2) *Integrating more narrow bitwidth multipliers within NBVEs (exploiting DLP within BLP) minimizes the cost of add-tree logic.* Encapsulating a larger number of narrow-bitwidth multipliers in an NBVE leads to amortizing the cost of add-tree logic across a wider array of multipliers and yields power/area efficiency. As it is shown in Figure 4, increasing L from 1 to 16 improves the power/area by $\approx 3\times$ in 1-bit slicing and $\approx 2.5\times$ in 2-bit slicing. However, this improvement in power/area gradually saturates. As such, increasing L beyond 16 does not provide further significant benefits.

(3) *The 2-bit slicing strikes a better balance between the complexity of the narrow-bitwidth multipliers and the cost of aggregation and operand delivery in CVUs.* 1-bit slicing requires 1-bit multipliers (merely AND gates), that also generates 1-bit values as the inputs of the adder-trees in NBVEs. However, slicing 8-bit operands to 1-bit require $8 \times 8 = 64$ NBVEs for a CVU, imposing a costly 64-input global adder-tree to CVUs. Consequently, as it is shown in Figure 4, 1-bit slicing does not provide any benefits compared to the conventional design. On the other hand, although 2-bit slicing results in generating 4-bit values by the multipliers as inputs to adder-trees in NBVEs, it quadratically decreases the total number of NBVEs in a CVU from 64 to 16. This quadratic decrease trumps using wider-bitwidth values as the inputs of adder-trees in NBVEs, significantly lowering add-tree cost. In conclusion, the optimal design choice for a digital CVU comes with 2-bit slicing and length of $L = 16$. Compared to a conventional digital design, this design point provides $2.0\times$ and $1.7\times$ improvement in power and area respectively, for an $8\text{-bit} \times 8\text{-bit}$ MAC operation. Note that 4-bit slicing provides lower power/area for CVU design, however, it leads to underutilization of compute resources when DNNs with less than 4-bits are being processed. As such, 2-bit slicing strikes a better balance between the efficiency of CVUs and their overall utilization.

(4) *Bit-parallel vector composability amortizes the cost of flexibility across the elements of vectors.* Prior bit-flexible works, both spatial (e.g., BitFusion [19]) and temporal (e.g., Stripes and Loom [10, 17]), enable supporting deep quantized DNNs with heterogeneous bitwidths at the cost of extra area overheads. BitFusion [19] exploits spatial bit-parallel composability for a scalar. This design can be assumed as one possible configuration of bit-parallel vector composability with 2-bit slicing and $L = 1$. As shown in Figure 4, this design point imposes 40% area overhead as compared to conventional design, while our design provides 40% reduction in area. Also our proposed CVU provides $2.4\times$ improvement in power as compared to Fusion Units in BitFusion. This result seems counter intuitive at the first glance as flexibility often comes with an overhead. In fact, the cost of bit-level flexibility stems from aggregation logic that puts the results back together. Our proposed bit-parallel

TABLE I:
EVALUATED DNN MODELS.

DNN Models	Type	Model Size (INT8)	Multiply-Adds (GOp/s)	Heterogenous Bitwidths
AlexNet	CNN	56.1 MB	2,678	First and last layer 8-bit, the rest 4-bit
Inception-v1	CNN	8.6 MB	1,860	First and last layer 8-bit, the rest 4-bit
ResNet-18	CNN	11.1 MB	4,269	First and last layer 8-bit, the rest 4-bit
ResNet-50	CNN	24.4 MB	8,030	All layers with 4-bit
RNN	RNN	16.0 MB	17	All layers with 4-bit
LSTM	RNN	12.3 MB	13	All layers with 4-bit

TABLE II:
EVALUATED HARDWARE PLATFORMS.

Chip	ASIC Platforms			GPU Platform	
	TPU-Like Baseline	BitFusion	BPVeC	Chip	RTX 2080 TI
# of MACs	512	448	1024	# of Tensor Cores	544
Architecture	Systolic	Systolic	Systolic	Architecture	Turing
On-chip memory	112 KB	112 KB	112 KB	Memory	11 GB (GDDR6)
Frequency	500 Mhz	500 Mhz	500 Mhz	Frequency	1545 Mhz
Technology Node	45 nm	45 nm	45 nm	Technology Node	12 nm

vector-level composability amortizes this cost across the elements of the vector. Moreover, since it reduces the complexity of the co-operating narrower bitwidth units, it leads to even further reduction.

C. Overall Architecture

Conceptually, bit-parallel vector composability is orthogonal to the architectural organization of the CVUs. We explore this design style using a 2D systolic array architecture, which is efficient for matrix-multiplications and convolutions, as explored by prior works [9]. In this architecture, called BPVEC, each CVU reads a vector of weights from its private scratchpad, while a vector of inputs is shared across columns of CVUs in each row of the 2D array. The scalar outputs of CVUs aggregate across columns of the array in a systolic fashion and accumulate using 64-bit registers.

IV. EVALUATION

A. Methodology

Workloads. Table I details the specification of the evaluated models. We evaluate our proposal using these neural models in two cases of homogenous and heterogeneous bitwidths. For the former one we use 8-bit datatypes for all the activations and weights and for the later we use the bitwidths reported in the results of the literature [4, 8, 12] that maintains the full-precision accuracy of the models.

ASIC baselines. For the experiments with homogeneous fixed bitwidths, we use a TPU-like accelerator with a systolic architecture. For the case of heterogeneous bitwidths, we use BitFusion [19], a state-of-the-art spatial bit-flexible DNN accelerator, as the comparison point. In all setups, we use 250 mW core power budget for all the baselines and the proposed accelerator in 45 nm technology node and with 500 Mhz frequency. Table II details the specifications of the evaluated platforms. We modify the open-source simulation

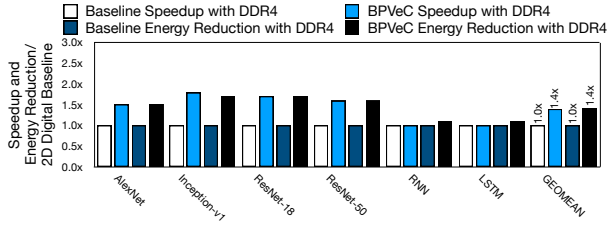


Fig. 5: Comparison to baseline; DDR4 memory and without bitwidth heterogeneity.

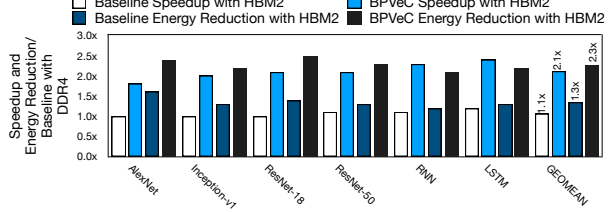


Fig. 6: Comparison to baseline; HBM2 memory and without bitwidth heterogeneity. infrastructure in [19] to obtain end-to-end performance and energy metrics for the TPU-like baseline accelerator, baseline BitFusion accelerator, as well as the proposed BPVEC accelerator.

GPU baseline. We also compare BPVEC to the Nvidia’s RTX 2080 TI GPU, equipped with tensor cores that are specialized for deep learning inference. Table II shows the architectural parameters of this GPU. For the sake of fairness, we use 8-bit execution for the case of homogenous bitwidths and 4-bit execution for heterogenous bitwidths using the Nvidia’s TensorRT 5.1 compiled with CUDA 10.1 and cuDNN 7.5.

Hardware measurements. We implement the proposed accelerator using Verilog RTL. We use Synopsis Design Compiler (L-2016.03-SP5) for synthesis and measuring energy/area. All the synthesis for the design space exploration presented in Figure 4 are performed in 45 nm technology node and 500 Mhz frequency and all the design points meet the frequency criteria. The on-chip scratchpads for ASIC designs are modeled with CACTI-P.

Off-chip memory. We evaluate our design style with both a moderate and high bandwidth off-chip memory system to assess its sensitivity to the off-chip bandwidth. For moderate bandwidth, we use DDR4 with 16 GB/sec bandwidth and 15 pJ/bit energy for data accesses. We model the high bandwidth memory based on HBM2 with 256 GB/sec bandwidth and 1.2 pJ/bit energy for data accesses [13].

B. Experimental Results

1) Without Bitwidth Heterogeneity

Figure 5 evaluates the performance and energy benefits of BPVEC across a range of DNNs with homogenous bitwidths (8-bit). The baseline uses the same systolic-array architecture as the BPVEC accelerator, but with conventional compute units that operate on individual operands and not bit-slices. Both designs use a DDR4 memory system. Bit-parallel vector composability enables our accelerator to integrate $\approx 2.0\times$ more compute resources compared to the baseline design under the same core power budget. On average, BPVEC provides 40% speedup and energy reduction. Across the evaluated workloads, CNN models enjoy more benefits compared to RNN ones. Unlike the CNNs that have significant data-reuse, the vector-matrix multiplications in RNNs have limited data-reuse and require extensive off-chip data accesses. As such, the limited bandwidth of the DDR4 memory leads to starvation of the copious on-chip compute resources in BPVEC.

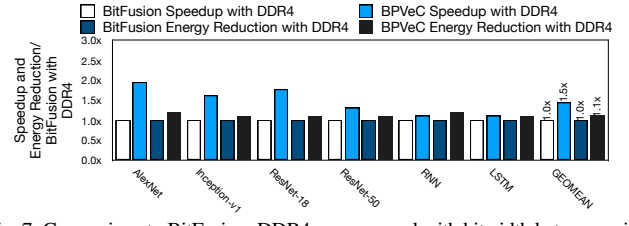


Fig. 7: Comparison to BitFusion; DDR4 memory and with bitwidth heterogeneity.

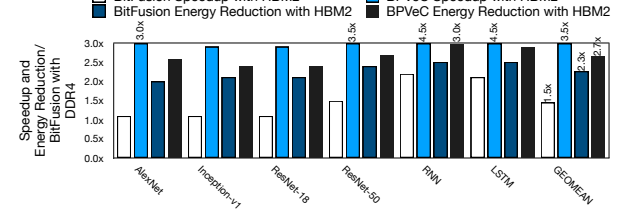


Fig. 8: Comparison to BitFusion; HBM2 memory and with bitwidth heterogeneity.

Figure 6 compares benefits from a high bandwidth memory (HBM2) for the baseline and BPVEC, normalized to the baseline with DDR4. While benefits from high bandwidth memory are limited for the baseline design, our BPVEC enjoys a $2.1\times$ and $2.3\times$ speedup and energy reduction, respectively. Results suggest that BPVEC is better able to exploit the increased bandwidth and reduced energy cost from HBM2 memory system. While all benchmarks see improved efficiency, benefits are highest for bandwidth-hungry RNN and LSTM.

2) With Bitwidth Heterogeneity

Figure 7 evaluates the performance and energy benefits for quantized DNNs with heterogenous bitwidths. The baseline in Figure 7 is BitFusion [19], a state-of-the-art accelerator that also supports flexible bitwidths, but at the scalar level. In this experiment, the baseline BitFusion and BPVEC use DDR4 memory. Bit-parallel vector composability enables our design to integrate $\approx 2.3\times$ more compute resources compared to BitFusion under the same core power budget. On average, BPVEC provides 50% speedup and 10% energy reduction over BitFusion. Across the evaluated workloads, CNN models enjoy more benefits compared to bandwidth-hungry RNN and LSTM.

Figure 8 studies the interplay of high off-chip bandwidth with flexible-bitwidth acceleration. The speedup and energy reduction numbers are normalized to BitFusion with moderate bandwidth DDR4. BPVEC provides $2.5\times$ speedup and 20% energy reduction over BitFusion with HBM2 memory ($3.5\times$ speedup and $2.7\times$ energy reduction over the baseline 2D BitFusion). RNN and LSTM, see the highest performance benefits ($4.5\times$), since these benchmarks can take advantage of both the increased compute units in BPVEC design, as well as the increased bandwidth from HBM2.

3) GPU Comparison

Figure 9 compares the Performance-per-Watt of BPVEC design with DDR4 and HBM2 memory and with homogenous (Figure 9 (a)) and heterogenous (Figure 9) bitwidths for DNN layers, respectively as compared to the Nvidia’s RTX 2080 TI GPU. With homogenous bitwidths (Figure 9 (a)), BPVEC achieves $33.7\times$ and $31.1\times$ improvements on average with DDR4 and HBM2 memory, respectively. Across the evaluated workloads, the RNN models see the most benefits. These models require a large amount of vector-matrix multiplications, which particularly suitable for the proposed bit-parallel vector composability design style. In the case of heterogenous bitwidths, the benefits go to $28.0\times$ and $29.8\times$

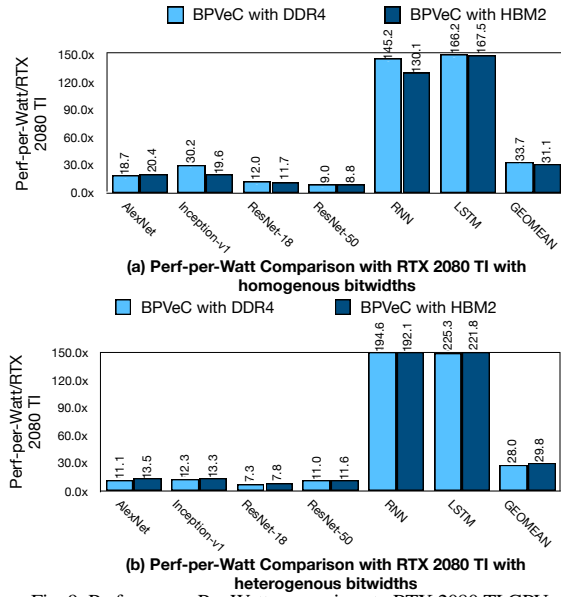


Fig. 9: Performance-Per-Watt comparison to RTX 2080 TI GPU.

with DDR4 and HBM2, respectively. The trends look similar to the homogenous 8-bit mode, since both the design points and the GPU baseline exploit the deep quantized mode of operations.

V. RELATED WORK

A large body of inspiring work has explored hardware acceleration for DNNs by exploiting their algorithmic properties such as data-level parallelism, tolerance for reduced precision and sparsification, and redundancy in computations. To realize the hardware accelerators, prior efforts have built upon isolated compute units that operate on all the bits of individual operands, and have used multiple compute units operating together to extract data-level parallelism. This work introduces a different design style that explores the interleaving of bit-level operations across compute units in the context of *vectors* and combine the benefits from bit-level parallelism and data-level parallelism, both of which are abundant in DNNs. Below, we discuss the most related works.

Design without support for bit-level composability. Prior works in TPU [9] and Eyeriss [2] design hardware accelerators to extract data-level parallelism in DNNs. SCNN [14], EIE [7], and Cnvlutin [1] use both zero-skipping and data-level parallelism for efficient DNN execution. Brainwave [6] also uses SIMD vectorized execution to extract data-level parallelism on FPGAs, however with fixed-bitwidth execution whose bitwidth is decided before synthesizing the FPGA on the design. ISAAC [16] and PRIME [3] build upon ResistiveRam(ReRam) technology to provide high energy efficiency. Further, ISAAC and PRIME operate on vectors of data in the analog (current) domain to mitigate the high cost of ADC. In contrast, we focus on interleaving of the bit-level and data-level parallelism in vector units, in addition to hardware support for bitwidth heterogeneity.

Design with support for bit-level flexibility through bit-serial computation. Stripes [10], Loom [17], UNPU [11] exploit the tolerance to reduced bitwidth in DNNs to yield performance benefits by exploring *bit-bit serial compute units*. The data-level parallelism compensates for bit-serial individual operations. Our design in contrast interleaves bit-level parallelism with data-level parallelism.

Designs with support for bit-level composability BitFusion [19] uses bit-level parallelism with a spatial design. We provide

a head-to-head comparison with BitFusion in Section IV-B2. Laconic [18] combines spatial bit-level composability with temporal execution to support for bit-sparsity and reduce the ineffectual computations. These inspiring efforts do not focus on bit-parallel vector composability that breaks the calculations across spatial composable units that cooperate at the level of vectors.

VI. CONCLUSION

Traditionally, neural accelerators have relied on extracting DLP using isolated and self-sufficient compute units that process all the bits of operands. This work introduced a different design style, *bit-parallel vector-composability*, that operates on operand bit-slices to interleave and combine the traditional data-level parallelism with bit-level parallelism. Across a range of deep models the results show that the proposed design style offers significant performance and efficiency compared to even bit-flexible accelerators.

VII. ACKNOWLEDGEMENT

This work was in part supported by generous gifts from Google, Qualcomm, Microsoft, Xilinx as well as the National Science Foundation (NSF) awards CNS#1703812, ECCS#1609823, CCF#1553192, Air Force Office of Scientific Research (AFOSR) Young Investigator Program (YIP) award #FA9550-17-1-0274, National Institute of Health (NIH) award #R01EB028350, and AirForce Research Laboratory (AFRL) and Defense Advanced Research Project Agency (DARPA) under agreement number #FA8650-20-2-7009 and #HR0011-18-C-0020. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes not withstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied of Google, Qualcomm, Microsoft, Xilinx, Samsung, NSF, AFOSR, NIH, AFRL, DARPA or the U.S. Government.

REFERENCES

- [1] J. Albericio *et al.*, “Cnvlutin: ineffectual-neuron-free deep neural network computing,” in *ISCA*, 2016.
- [2] Y.-H. Chen *et al.*, “Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks,” in *ISCA*, 2016.
- [3] P. Chi *et al.*, “Prime: A novel processing-in-memory architecture for neural network computation in rram-based main memory,” in *ISCA*, 2016.
- [4] J. Choi *et al.*, “Pact: Parameterized clipping activation for quantized neural networks,” *arXiv preprint arXiv:1805.06085*, 2018.
- [5] A. T. Elthakeb *et al.*, “Relec: an automatic reinforcement learning approach for deep quantization of neural networks,” *arXiv preprint arXiv:1811.01704*, 2018.
- [6] J. Fowers *et al.*, “A configurable cloud-scale dnn processor for real-time ai,” in *ISCA*, 2018.
- [7] S. Han *et al.*, “Eie: efficient inference engine on compressed deep neural network,” in *ISCA*, 2016.
- [8] I. Hubara *et al.*, “Quantized neural networks: Training neural networks with low precision weights and activations,” *arXiv*, 2016.
- [9] N. P. Jouppi *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *ISCA*, 2017.
- [10] P. Judd *et al.*, “Stripes: Bit-serial deep neural network computing,” in *MICRO*, 2016.
- [11] J. Lee *et al.*, “Unpu: A 50.6 tops/w unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision,” in *ISSCC*, 2018.
- [12] A. K. Mishra *et al.*, “WRPN: wide reduced-precision networks,” *arXiv*, 2017.
- [13] M. O’Connor *et al.*, “Fine-grained dram: energy-efficient dram for extreme bandwidth systems,” in *MICRO*. IEEE, 2017, pp. 41–54.
- [14] A. Parashar *et al.*, “SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks,” in *ISCA*, 2017.
- [15] M. Samragh *et al.*, “Codex: Bit-flexible encoding for streaming-based fpga acceleration of dnn,” *arXiv preprint arXiv:1901.05582*, 2019.
- [16] A. Shafiee *et al.*, “Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars,” in *ISCA*, 2016.
- [17] S. Sharify *et al.*, “Loom: Exploiting weight and activation precisions to accelerate convolutional neural networks,” *arXiv*, 2017.
- [18] S. Sharify *et al.*, “Laconic deep learning inference acceleration,” in *ISCA*, 2019.
- [19] H. Sharma *et al.*, “Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network,” in *ISCA*. IEEE, 2018, pp. 764–775.