An ASIC Accelerator for QNN With Variable Precision and Tunable Energy Efficiency

Ankit Wagle, *Member, IEEE*, Gian Singh, *Member, IEEE*, Sunil Khatri, *Senior Member, IEEE*, and Sarma Vrudhula, *Life Fellow, IEEE*

***Abstract*—This article presents TULIP, a new architecture for a variable precision quantized neural network (QNN) inference. It is designed with the goal of maximizing energy efficiency per classification. TULIP is constructed by arranging a collection of *unique* processing elements (TULIP-PEs) in a single-instruction–multiple-data (SIMD) fashion. Each TULIP-PE contains *binary neurons* that are interconnected using multiplexers. Each neuron also has a small dedicated local register connected to it. The binary neurons are implemented as standard cells and used for implementing threshold functions, i.e., an inner-product and thresholding operation on its binary inputs. The neurons can be reconfigured with a single change in the control signals to implement all the standard operations used in a QNN. This article presents novel algorithms for implementing the operations of a QNN on the TULIP-PEs in the form of a schedule of threshold functions. TULIP was implemented as an ASIC in TSMC 40nm-LP technology. A QNN accelerator that employs a conventional multiply and accumulate-based arithmetic processor was also implemented in the same technology to provide a fair comparison. The results show that TULIP is 30×–50× more energy-efficient than an equivalent design, without any penalty in performance, area, or accuracy. Furthermore, TULIP achieves these improvements without using traditional techniques such as voltage scaling or approximate computing. Finally, this article also demonstrates how the run-time tradeoff between accuracy and energy efficiency is done on the TULIP architecture.**

***Index Terms*—Area efficient, energy efficient, high performance, high throughput, quantized neural network (QNN), reconfigurable, threshold logic.**

I. INTRODUCTION

DEEP neural networks (DNNs) have been remarkably successful in numerous applications of pattern recognition and data mining, including speech recognition, image classification, object recognition, autonomous vehicles and robotics, recommendation systems, and many more. Consequently, they have become the dominant algorithmic framework in machine learning. DNNs are computationally and *energetically* intensive algorithms that perform billions of floating-point multiply–accumulate operations on very large dimensional datasets, some involving tens of billions of parameters [1]. Because *training* of large networks entails much greater computational effort and storage than inference, it is performed on high-performance servers with numerous CPU and GPU cores.

The energy cost and the environmental impact of training and inference of large DNNs are fast becoming unsustainable. For instance, training of the GPT-3 model with 175B parameters using NVidia's A100 with 1024 GPUs would consume 936 MWh of energy and take 34 days at a cost of $4.6M. Models even larger than the GPT-3 are being developed [1].

Improvements in the energy efficiency of DNNs are not just limited to high-performance servers or desktop machines. The latest *midrange* and *high-end* mobile SoCs [2], [3] are being equipped with custom NN hardware accelerators to perform inference on mobile (e.g., mostly smartphones) and edge devices (e.g., IoT devices deployed in numerous spaces) for many of the above applications. The energy efficiency of inference on battery-powered devices is also of critical importance in terms of value to the customer and environmental impact. Given the rapid proliferation of ML techniques, several orders of magnitude improvement in energy efficiency over CPU–GPU implementations for training and inference of DNNs is needed for ML technology to be sustainable.

FPGA and ASICs are the two alternates to CPU–GPU implementations. The energy efficiency and throughtput of FPGA implementations of DNNs is in between ASICs and CPU–GPUs [4], [5]. Past and ongoing works on executing DNNs on FPGAs include the development of optimizing compilers that automatically map DNNs expressed in standard frameworks onto FPGAs with the objective of minimizing latency or throughput subject to constraints on energy, memory bandwidth, the number of DSPs [6], [7], [8], [9], [10].

ASIC implementations have orders of magnitude higher energy efficiency than the CPU–GPU implementations. Analog and mixed-signal solutions implement the inner product of fixed-weight matrices and input vectors by summing currents in crossbar arrays, where the weights are realized by various types of resistive elements (ReRAM [11], MTJ [12], and Flash [13]). This approach continues to be an active area of on-going research, driven by the constant introduction of novel nonvolatile multistate memory devices.

Manuscript received 8 July 2023; revised 26 October 2023 and 7 January 2024; accepted 9 January 2024. Date of publication 24 January 2024; date of current version 20 June 2024. This work was supported in part by the National Science Foundation under Grant 2008244, and in part by Qualcomm Technologies Inc. This article was recommended by Associate Editor T. Theocharides. *(Corresponding author: Sarma Vrudhula.)*

Ankit Wagle, Gian Singh, and Sarma Vrudhula are with the School of Computing and Augmented Intelligence, Arizona State University, Tempe, AZ 85281 USA (e-mail: awagle1@asu.edu; gsingh58@asu.edu; vrudhula@asu.edu).

Sunil Khatri is with the Department of Electrical and Computer Engineering, Texas A&M University at College Station, College Station, TX 77843 USA (e-mail: sunilkhatri@tamu.edu).

Digital Object Identifier 10.1109/TCAD.2024.3357597

Purely digital ASIC implementations are constructed by synthesis of custom logic blocks for specific operations, such as 2-D convolution, inner product, matrix multiplication, and others [14], [15], each optimized for throughput and energy efficiency. A common approach to maximizing energy efficiency is to pare down the functionality of the circuit, e.g., eliminate processing of the integer layers as in XNORBIN [16], or reduce the size of the kernels and/or reduce the bitwidth of operands as in [17] and [18].

A few architectures, however, support complete end-to-end neural networks, i.e., convolution layers and fully connected layers, such as YodaNN [14] (which supports 12-bit inputs and 1-bit weights), UNPU [19] (which supports 1–16 bit variable precision inputs and weights), and BitBlade [20] (which supports bit precision of 2, 4, and 8 for inputs and weights). They also support variable-sized kernels. The UNPU [19] architecture is based on bit-serial processing of the weights with the activations to compute the partial products. It uses lookup-table-based processing elements (LBPEs) and the overall architecture is designed to perform dense matrix multiplications with high parallelism. The independent DNN cores in the UNPU use a *fixed-size* accumulator to obtain partial sums which are finally processed in a separate 1-D single-instruction–multiple-data (SIMD) core. The SIMD core performs vector operations such as nonlinear activation or element-wise multiplication to generate the final output. The UNPU includes a RISC controller to orchestrate the intercore communication via an NoC during the DNN operation.

The processing element (PE) in BitBlade [20], introduces a new bitwise summation scheme which reduces the shift and add logic in the PE to reduce overall area and power consumption. The PE of BitBlade consists of 16 2-bit multipliers and summation logic to perform the multiply and accumulate (MAC) operation. The operands are decomposed into chunks of 2-bits to utilize 2-bit multipliers. The other operations of the quantized neural network (QNN) such as linear activation is performed using either additional dedicated logic or by using the host CPU.

YodaNN [14] is an SIMD processor that consists of an array of MAC PEs with on-chip standard-cell memory (that can be synthesized). The design presented in this article, named TULIP, is also a complete end-to-end system. A detailed comparison of YodaNN and TULIP is presented in Section VIII.

Regardless of whether it is an FPGA or ASIC implementation, throughput and energy efficiency can also be improved by modifying the structure of the NN. This includes tuning the hyperparameters [21], [22], modifying the network structure by removing the weights and connections [23], [24], or by altering the degree of quantization [25], [26]. Another category of methods focuses on reducing the huge energy expenditure for moving data between the processor and off-chip memory, which is especially acute in NNs because of the large number of weights involved. The techniques to mitigate this include maximizing the reuse of data fetched from memory [27], [28], or transferring compressed data from the memory to the processor [29], [30].

Of the many available techniques for modifying NN structure, quantization remains the best way to achieve high energy efficiency and reduce computation time [31], [32], especially for energy-constrained systems. Quantization refers to using smaller bit-widths for the weights and/or the inputs during training, reducing them from 32-bit values to anywhere from 8-bit to 1-bit values. The term BNN refers to neural networks with 1-bit weights and inputs. Anything larger than that, but below full 32-bit precision is referred to as QNN. Quantization takes advantage of the fact that the accuracy of NNs is not very sensitive to substantial reductions in bit-widths until some critical value. Depending on the network, 4-bit to 1-bit QNNs for mobile applications provide an excellent tradeoff between energy efficiency and throughput versus accuracy [32].

II. OVERVIEW OF THIS ARTICLE

This article presents the design of an ASIC for accelerating QNNs. The design, named TULIP, achieves substantial improvements in energy efficiency compared to the state-of-the-art design of QNNs [14]. Energy efficiency is defined as *throughput-per-watt*, or equivalently, *operations-per-joule*.

Fig. 1 shows the main components of TULIP. The following is a summary of these components, which will be elaborated upon in the subsequent sections.

1) Fig. 1(a) shows the top-level system diagram of TULIP. It is a scalable SIMD machine that consists of a collection of independent, concurrently executing TULIP *processing elements* (TULIP-PEs), shown in Fig. 1(b). The architecture of the TULIP-PE is very different from the PEs used in any other QNN accelerators [11], [14], [18], [33]. It consists of a small network of *binary neurons*, whose circuit structure is shown in Fig. 1(c), and described in greater detail in Section III.
2) Briefly, a neuron is a *clocked logic cell* that computes a threshold function of its inputs, on a clock edge. It is a *mixed-signal* circuit, whose inputs and outputs are logic signals but internally it computes the inner-product and threshold operation of a neuron, i.e., $f(x_1, \ldots, x_n | w_1, \ldots, w_n, T) = \sum_i^n w_i x_i \geq T$. Implemented as a standard cell, and after optimized for robustness and accounting for process variations, a neuron in 40nm is just a little larger than a conventional D-type flip-flop [34]. The neurons in a TULIP-PE can be configured at run-time to execute all the operations of a QNN, namely the accumulation of partial sums, comparison, max-pooling, and RELU. Consequently, only a single PE is required to implement all the operations in a QNN, and switching between operations is accomplished by supplying an appropriate set of logic signals to its inputs, which incurs no extra overhead in terms of area, power, or delay.
3) Unlike conventional MAC [14] or fixed-size accumulator-based [19] PEs, that are designed to operate at maximum bit-width (determined at design-time), the bit precision of TULIP-PEs can be changed within a single cycle without incurring a delay or energy penalty. The TULIP-PEs enable control over the precision of both

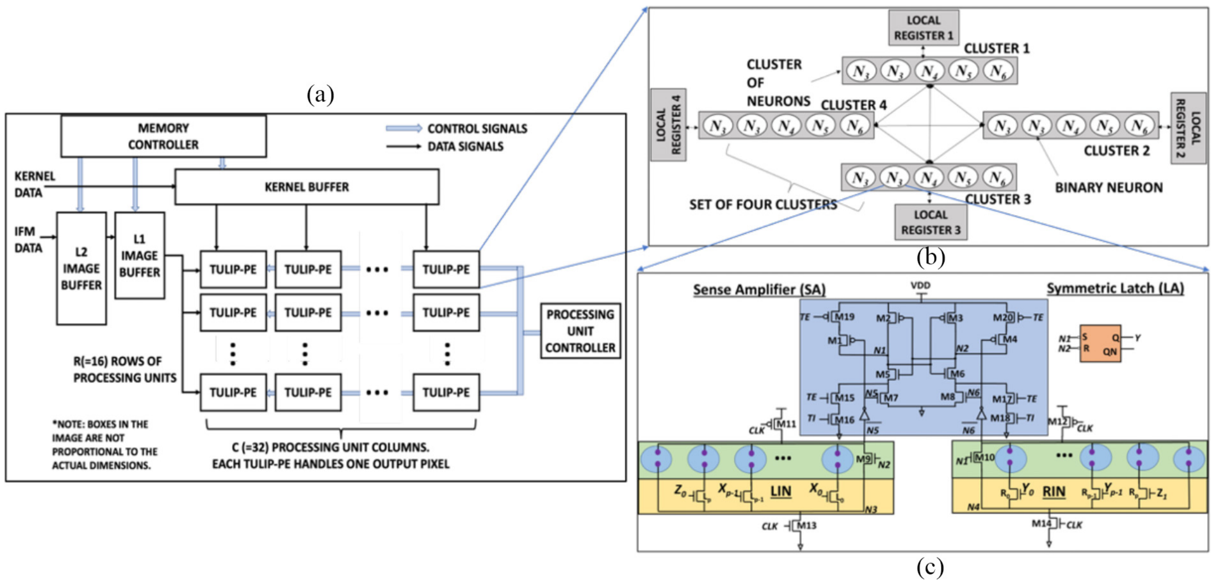


Fig. 1. TULIP architecture overview. (a) Top-level architecture of TULIP: controller configures the processing units. The input pixels and weights are sent through image and kernel buffers. The output of the processing units is collected in the output buffers before sending it back to the memory. (b) Architecture of a TULIP-PE, consisting of four clusters and four local registers. Each cluster contains K neurons (K=5). (c) Architecture of a binary neuron.

inputs (weights and activation) and output unlike [19]. The operation of TULIP-PEs prevents overprovisioning of the hardware for an operation of a certain bitwidth, thus improving the energy efficiency of the overall computation. This characteristic allows for making tradeoffs between energy efficiency and accuracy at run-time.

4) Against the state-of-the-art MAC units used in QNN accelerators [14], the TULIP-PE is $\approx 16\times$ smaller and consumes $125\times$ less power. Although it is $9.6\times$ slower, this can be compensated by replicating 16 PEs and operating TULIP in an SIMD mode, executing multiple workloads in parallel that share inputs, which reduces the need to repeatedly fetch data from off-chip memory.
5) Since the neurons in the TULIP-PE have limited fan-in, much larger inner product calculations have to first be decomposed into smaller bit-width operations and then scheduled on the TULIP-PEs. For this, a novel *routing-aware*, resource-constrained, scheduling algorithm is presented that maps the nodes of a QNN onto TULIP-PEs.
6) The combined effect of the low-area of TULIP-PE, the uniform computation at the individual node and network levels, and the mapping algorithm results in an improvement of up to $50\times$ in energy efficiency for QNNs over a MAC-based design for the same area and performance.

This article is organized as follows. Sections III, IV, and VI describe the architecture of the binary neuron, TULIP-PE, and the top-level architecture of TULIP, respectively. Section V presents the scheduling algorithm needed to execute each node of the QNN on a TULIP-PE. Section VII then describes how the small size of the TULIP-PEs enables us to deploy a number of them in the same space as a conventional processing unit, thereby enabling better weight reuse. Finally, Section VIII presents a both quantitative and qualitative evaluation of TULIP-PEs and the TULIP architecture against equivalent state-of-the-art architectures.

Note that we presented a preliminary version of this work in [35]. This article includes an updated hardware architecture that extends support for varying precision of QNNs while also significantly improving the overall energy efficiency. We perform an extensive evaluation using multiple NNs and datasets to demonstrate the efficacy of the updated TULIP. This article also provides a generalized formulation for mapping arbitrary compute graphs to the TULIP-PEs.

III. BACKGROUND

A Boolean function $f(x_1, x_2, \ldots, x_n)$ is called a threshold function if there exist weights w_i for $i = 1, 2, \ldots, n$ and a threshold T[1] such that

$$f(x_1, x_2, \ldots, x_n) = 1 \Leftrightarrow \sum_{i=1}^{n} w_i x_i \geq T \tag{1}$$

where $\sum$ denotes the arithmetic sum. A threshold function is denoted by the pair $(W, T) = [w_1, w_2, \ldots, w_n; T]$. An example of a threshold function is $f(x_1, x_2, x_3, x_4) = x_1x_2 \vee x_1x_3 \vee x_1x_4 \vee x_2x_3x_4$, with $[w_1, w_2, w_3, w_4; T] = [2, 1, 1, 1; 3]$.

A binary neuron is a circuit that realizes a threshold function defined by (1). Fig. 1(c) shows the design of the binary neuron that is used in TULIP. A detailed description of its operation, the algorithms for optimizing its robustness, performance, power, and area, and its use in ASIC synthesis appear in [34]. As the design of the binary neuron is not the focus of this article, only a summary of its operation is presented here.

[1]W.L.O.G., the weights w_i and threshold T can be integers [36].

The binary neuron shown in Fig. 1(c) has four main components[2]: 1) the left input network (LIN); 2) the right input network (RIN); 3) a sense amplifier (SA); and 4) an output latch (LA). The SA outputs are differential digital signals $(N1, N2)$, with $(1, 0)$ and $(0, 1)$ setting and resetting the latch. The LIN and RIN consist of a set of branches, each branch consisting of two devices in series, one (labeled Z) which provides a *configurable* conductance between its two terminals, and a MOSFET driven by an input signal x_i. The conductance of a branch controlled by x_i serves as a proxy of the weight w_i in (1). Let $G_L(X|W)$ and $G_R(X|W)$ denote the conductance of the LIN and RIN, respectively. For a given threshold function f, the conductance of each branch is configured so $G_L(X|W) > G_R(X|W)$ for all on-set minterms of f, and vice versa for all off-set minterms of f.

When CLK $= 0$, the LIN and RIN play no role and $(N1, N2) = (1, 1)$, and the output Y of the latch remains unchanged. Before the clock rises, inputs are applied to the LIN and RIN. Suppose that an on-set minterm is applied. When CLK $0 \to 1$, both $N1$ and $N1$ will start to discharge. However, since $G_L(X|W) > G_R(X|W)$, $N1$ will discharge much faster than $N2$, which will also turn off the discharge of $N2$, resulting in $N2$ going back to 1. The result is $(N1, N2) = (0, 1)$, which will set the latch output $Y = 1$. Thus, the binary neuron in Fig. 1(c) may be viewed as a multi-input, edge-triggered *flip-flop* that computes a threshold function of its inputs on a clock edge. Note that there are a number of choices for realizing the configurable conductance devices, which are explored in [34], [37], and [38].

IV. TULIP-PE IMPLEMENTATION

A. Primitive Operations

The TULIP-PE is designed to implement the nodes of all the layers in a QNN. This is achieved by decomposing the node's operations (multiplication, ReLU, etc.) into K-bit primitive operations. These are addition, comparison, or logic operations that are executed in at-most two cycles. They are realized as *threshold functions* and computed by *artificial neurons*. N-bit ($N > K$) operations are executed as a sequence of K-bit operations. In this section, we describe the representation of the primitive operations threshold functions. The following notation will be used to describe single and multibit values.

1) Characters (e.g., A or A_0, etc.) without dimensions specified will denote variables that may either be a single-bit or a multibit value.
2) Square brackets (e.g., $A_{[0]}$, $A_{[K-1:0]}$, etc.) are used to represent bit vectors.
3) Characters having subscripts but no square brackets (e.g., A_0) denote single-bit variables.
4) Bit replication is denoted with the variable enclosed in curly braces with the multiplier in the subscript. For instance, $\{A_{[0]}\}_{\times N}$ represents an N-bit vector with all bits equal to $A_{[0]}$.

Equation (2) shows a template that is used to describe the primitive operations using threshold functions. In the expression, p is an integer, X and Y are p-bit operands, and Z_0 and Z_1 are 1-bit values

$$Q(p, Z_0, X, Z_1, Y) = Z_0 + \sum_{j=0}^{p-1} 2^j X_j \geq Z_1 + \sum_{j=0}^{p-1} 2^j Y_j. \quad (2)$$

1) Logic Operations: Primitive logic functions AND, OR, and NOT are threshold functions [36]. The corresponding logic operations on K-input operands A and B are denoted as $LK(A, B)$ (binary) or $LK(A)$ (unary). They are realized as a vector of K threshold functions on each corresponding bit

$$\begin{aligned}&\text{AND}\big(A_{[K-1:0]}, B_{[K-1:0]}\big)\\ &\quad = \big[Q\big(1, 0, A_{[K-1]}, 1, \overline{B}_{[K-1]}\big), \ldots, Q\big(1, 0, A_{[0]}, 1, \overline{B}_{[0]}\big)\big]. \quad (3)\end{aligned}$$

As an example, consider a 2-bit AND operation between two 1-bit operands A and B, which can be calculated using $Q(1, 0, A, 1, \overline{B})$. By substituting appropriate values in (2), we get $0+A \geq 1+\overline{B}$, which in turn can be rewritten as $A+B \geq 2$.

Other K-bit logic operations are similarly defined. These can be computed in one cycle by a neuron cluster in an NPE. On the other hand, XOR$(A_{[i]}, B_{[i]})$ is realized as a two-level threshold network and therefore requires two cycles. In terms of Q, it is derived as follows: XOR operation is represented as a pseudo-Boolean equation $A_{[i]}+B_{[i]}-2A_{[i]}B_{[i]}$. This can be written in the form of an inequality $A_{[i]} + B_{[i]} - 2A_{[i]}B_{[i]} \geq 1$ which in-turn can be written as $A_{[i]} + B_{[i]} \geq 1 + 2A_{[i]}B_{[i]}$. Consequently, by using the representation in (2), and substituting the term AB with (3), we get

$$\begin{aligned}&\text{XOR}\big(A_{[i]}, B_{[i]}\big)\\ &\quad = Q\big(2, A_{[i]}, \{0, B_{[i]}\}, 1, \{Q\big(1, 0, A_{[i]}, 1, \overline{B}_{[i]}\big), 0\}\big). \quad (4)\end{aligned}$$

For instance, an XOR operation between two 1-bit operands A and B can be rewritten using a combination of (4) and (2) as $A + B - 2AB \geq 1$.

2) Addition ($ADDK(A, B, C_0)$): Let C_{i+1} denote the carryout of stage i, $i \geq 0$. A carry *lookahead* of size i means that C_i is expressed as a function of $A_{[i-1:0]}$, $B_{[i-1:0]}$, and C_0. While the carryout function is a threshold function regardless of the size of the lookahead, the sum function S_i is a threshold function of carry-out C_{i+1} and carry-in C_i, as shown in Fig. 2. Hence, a K-bit addition, denoted by $ADDK$, takes two cycles. C_{i+1} and S_i are expressed as

$$C_{i+1} = Q\big(i+1, C_0, A_{[i:0]}, 1, \overline{B}_{[i:0]}\big) \quad 0 \leq i \leq K-1 \quad (5)$$
$$S_i = Q\big(2, A_{[i]}, \{0, B_{[i]}\}, 0, \{C_{i+1}, \overline{C}_i\}\big). \quad (6)$$

To illustrate, consider an addition operation involving three 1-bit operands A, B, and C_0. This operation can be computed using (5) and (6). When we substitute the appropriate values into (2), we obtain the following.

1) The carry bit C_1 can be expressed as $C_0 + A \geq 1 + \overline{B}$. This can be further rewritten as $A + B + C_0 \geq 2$.
2) The sum bit S_0 can be represented as $A+B \geq 2C_1 + \overline{C_0}$. This, in turn, can be rewritten as $A+B+C_0-2C_1 \geq 1$.

3) Comparison ($COMPK(A_{[K-1:0]}, B_{[K-1:0]}, C)$): This computes the predicate $Y = (A_{[K-1:0]} + C > B_{[K-1:0]})$. In terms of Q, it is represented by

$$Y = Q\big(K, C, A_{[K-1:0]}, 1, B_{[K-1:0]}\big). \quad (7)$$

[2]This is a simplified version of the design shown in [34], with the test and *programming* circuitry not shown.

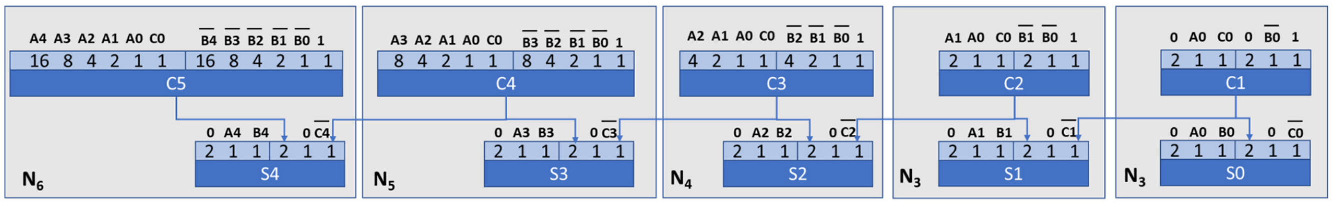


Fig. 2. 5-bit carry lookahead adder using binary neurons that adds two 5-bit numbers A and B, and a 1-bit carry-in $C0$. Each box represents (2), such that the left sub-box and right sub-box represent the left- and right-hand side of the equation, respectively.

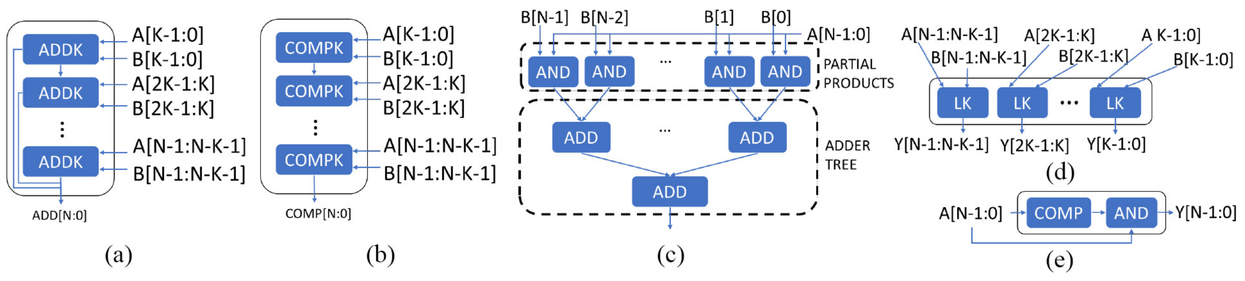


Fig. 3. Decomposition of various N-bit operations needed for QNNs into K-bit primitive operations. Here, $K < N$. (a) ADD. (b) COMP. (c) MUL. (d) AND/OR/NAND/NOR.

For N-bit operands ($N > K$), addition, comparison, logic, multiplication, and ReLU operations (among other operations) can be realized using K-bit primitive operations. Examples are shown in Fig. 3. These primitive operations can be executed sequentially on a TULIP-PE.

B. Hardware Architecture of TULIP-PE

A TULIP-PE [Fig. 1(b)] contains four clusters, each cluster containing K neurons. The neurons in each cluster are labeled $\mathcal{N}_\kappa$, where κ is the fan-in of the neuron in a cluster (indexed left to right). The ith significant bit ($i \in [1, K]$) of a primitive operation is computed by the ith neuron of a cluster. Therefore, the fan-in needed for a cluster's ith neuron is determined by the maximum number of inputs needed to represent the threshold function corresponding to the ith bit of every primitive operation.

As shown in Fig. 1(b), multiplexers are used to connect each neuron to its external inputs, to its neighboring neurons, its local registers (designed using latches), and to its own output (feedback). In the present implementation of the TULIP-PE, the weights associated with the neurons are chosen so to allow the implementation of all the primitive operations by simply applying the appropriate signals to each neuron's inputs, and also to ensure that neuron $\mathcal{N}_i$ can realize all the functions realizable by $\mathcal{N}_j, j < i$.

TULIP-PE requires a minimum of four clusters to ensure a single cycle delay between the launch of any two consecutive primitive operations. Considering that each primitive operation can be represented as a two-level (or one-level) computation of threshold functions, only two clusters are needed to perform the computation at any given time (compute mode), while the remaining two clusters are needed to read operands from their respective local registers and share them with the first two clusters (routing mode). The clusters switch between the compute and routing modes depending on the local registers in which the operands are stored and the local register to which the output must be written.

Note that the number of bits that can be processed in each cycle increases with the number of neurons K in each cluster. The larger the K, the better the performance. However, as K increases, the maximum fan-in of the binary neurons in each cluster also increases. Since there is a maximum fan-in limitation of the binary neuron [34], in the present implementation of TULIP-PE, $K = 5$.

V. REALIZING QNN NODE ON TULIP-PE

A QNN is a directed acyclic graph (DAG), where each node either represents an inner product that involves a sum of multibit products, or a nonlinear activation function (e.g., ReLU, etc.). The multibit products are computed using multibit logic and addition operations (see Fig. 3), which are primitive operations that are performed by a network of neurons. Thus, at the lowest level of granularity, a QNN node is a network of threshold functions that must be scheduled on the neurons (the compute elements) in the TULIP-PE with the objective of minimizing the completion time, subject to the registers and the routing constraints.

The threshold graph scheduling (TGS) problem is the same as the well-studied problem of mapping a dataflow graph (DFG) of computations onto a course grain reconfigurable array (CGRA). There is an extensive body of literature on CGRA architectures and scheduling computations onto them that spans more than two decades. A precise formulation of the CGRA scheduling problem first appeared in [39] and was shown to be NP-complete. In the Appendix, we present a precise formulation of TGS, which is the problem of scheduling a compute graph of threshold functions onto a specific network of neurons that constitute a TULIP-PE.

Since existing approaches to solve the above problem have exponential time complexity for the number of nodes in the

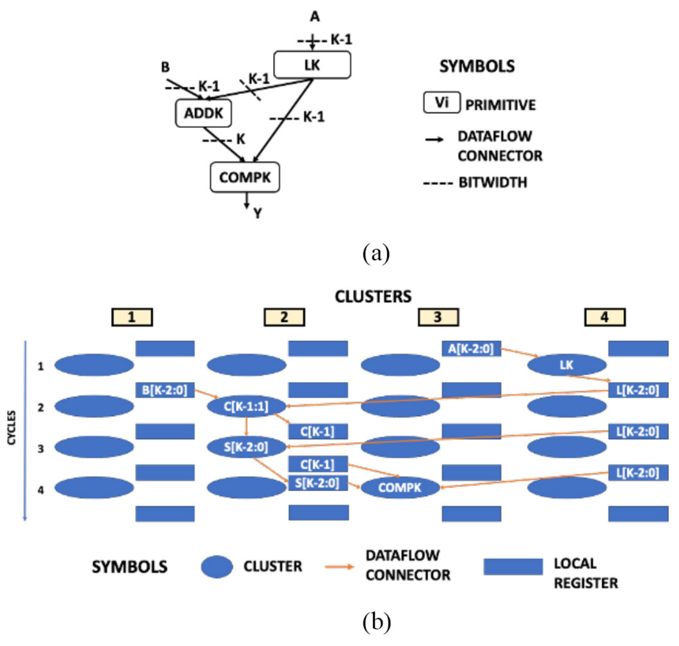


Fig. 4. Mapping a primitive graph G_P to a resource graph, where each resource is either a cluster or local register. (a) Primitive graph G_P. (b) Mapping to resource graph.

compute and resource graphs, they do not scale well. In the following, we present an alternate approach that is efficient and scalable. This is done by increasing the granularity of the nodes in the compute and resource graphs, which results in a drastic reduction in their sizes. The nodes in the compute graph are now primitive operations and the compute units are now clusters. The mapping problem is further simplified because a new operation can be initiated on a cluster on every cycle, i.e., its initiation interval is one. We first compute the register-aware, minimum latency schedule of the primitive graph on the clusters and then configure the neurons on each cluster to compute the function of the primitive node assigned to each cluster. This is illustrated in Fig. 4, which shows a feasible mapping of a primitive graph to a resource graph. The compute graph in Fig. 4(a) contains three primitive operations LK, ADDK, and COMPK, which are initialized in consecutive cycles as shown in Fig. 4(b). Operands A and B are stored in local registers 3 and 1. The operation LK is executed in cluster 4 and stored in its local register. The sum and carry bits of ADDK operation are calculated in cluster 2 using the data stored in local registers 1 and 4 and the result is stored in local register 2. Finally, the data from local registers 2 and 4 are used to compute COMPK to generate the final output Y.

A. Scheduling Primitive Graph on TULIP-PE

Definition 1 (Primitive Graph $G_P(V_P, E_P)$): This is a DAG where each node $v \in V_P$ represents a K-input primitive operation, i.e., K-bit addition, comparison, or logic. Each edge $e \in E_P$ represents a data dependency between the primitive operations.

An integer linear programming (ILP) formulation is presented for the problem of scheduling a primitive graph that represents a single QNN node, on a TULIP-PE. The principle behind the design of ILP is based on the high-level scheduling algorithm presented in [40], but differs from it because of the unique register and data routing constraints and the fact that the initiation interval of a cluster is one. Table I shows the notation used in the ILP formulation.

TABLE I
NOTATION FOR ILP USED TO SOLVE THE PRIMITIVE SCHEDULING PROBLEM

Notation	Description
(s_v, e_v)	Lifetime of storage of node v.
d_u	Delay of a primitive operation ($1 \leq d_u \leq 2$) cycles.
b_u	Bits needed to store the output of a primitive operation.
$\chi_{v,r,t}$	Value is 1 if output of node v is stored in register r at time t.
$\rho_{v,r}$	Value is 1 if output of node v is stored in register r at any time.
$\tau_{v,t}$	Value is 1 if output of node v is available at time t in any of the local registers.
$u \prec v$	u is the immediate predecessor of v.
E	Makespan (execution time) of G_P on TULIP-PE.

The primitive scheduling problem has to establish bindings between operations v, time steps t, and resources (local registers) r, since clusters store outputs in their respective local registers. Such bindings are represented using triple-indexed binary decision variables $\chi_{v,r,t}$ shown in

$$\chi_{v,r,t} = \begin{cases} 1, & \text{if } v \text{ is mapped to } r \text{ at time } t \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

Using the above equation, two additional binding variables are derived: $\rho_{v,r}$, which represents the mapping of v with local register r, and $\tau_{v,t}$ which represents the mapping of v with time t. These variables are used to express resource and time-specific constraints, respectively

$$\rho_{v,r} = \bigvee_{t=0}^{t=T-1} \chi_{v,r,t} \tag{9}$$

$$\tau_{v,t} = \bigvee_{r=0}^{r=3} \chi_{v,r,t}. \tag{10}$$

There are L local registers, each of size B bits. The minimum time required to execute all the primitives on G_R is $T = 2|V_P|$. With the goal of minimizing the makespan E (execution time) of G_P on TULIP-PE, the following constraints are needed to define the set of feasible solutions:

$$\textbf{Minimize } E \text{ such that.} \tag{11}$$

1) Resource Availability Constraints: These constraints are added to ensure that the local registers are not overutilized. The first constraint (12) ensures that the storage used by a local register r at any time t must never exceed the maximum capacity B

$$\sum_{\forall v \in V_P} b_v \cdot \chi_{v,r,t} \leq B \ \forall r \in [0, 3] \ \forall t \in [0, T-1]. \tag{12}$$

The second constraint (13) ensures that each primitive's output is stored in only one local register

$$\sum_{\forall r \in J} \rho_{v,r} = 1 \ \forall v \in V_P. \tag{13}$$

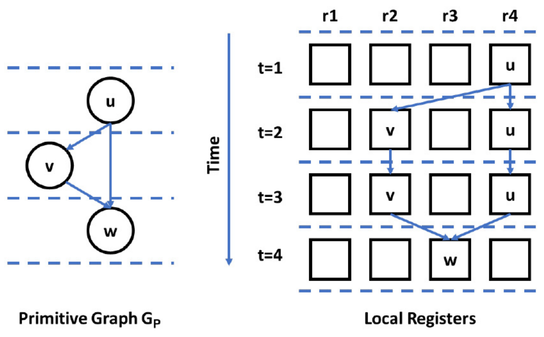


Fig. 5. Example to illustrate routing constraints in the primitive scheduling problem. The output of each node in the primitive graph G_P is stored in the local registers of TULIP-PE.

TABLE II
NUMBER OF ILP DECISION VARIABLES AND THE RUN-TIME REQUIRED TO GENERATE SCHEDULE ON TULIP-PE. HERE, THE COMPUTE GRAPH IS A NEURON THAT COMPUTES $\sum_{i=0}^{N-1} w_i x_i$ FOR VARYING N

Neuron Inputs (N)	Decision Variables	Time (Sec)
64	6.88E+03	0.04
128	2.81E+04	0.12
256	1.14E+05	0.49
512	4.56E+05	2.14
1024	1.83E+06	9.32
2048	7.33E+06	41.82
4096	2.93E+07	216.46

2) Precedence Constraints: Constraint in (14) is added to ensure that the data dependency due to the precedence relationship between any two primitives u and v is satisfied in the schedule

$$s_u + d_u \leq s_v \leq e_u + 1 \ \forall u, v \in V_P, \quad u \prec v. \tag{14}$$

For the schedule in Fig. 5, $s_u = 1$, $e_u = 3$, $d_u = 1$, $s_v = 2$.

3) Timing Validity Constraints: These constraints ensure that the start and end times of all the nodes are valid and feasible (15), and that the start times of any two nodes are not equal (16)

$$0 \leq s_v \leq e_v \leq T - 1 \ \forall v \in V_P \tag{15}$$

$$s_u \neq s_v \ \forall u, v \in V_P, \quad u \neq v \tag{16}$$

$$\tau_{v,t} = \begin{cases} 1, \ s_v \leq t \leq e_v \\ 0, \ \text{otherwise.} \end{cases} \tag{17}$$

Constraint in (18) is added to identify the end time of the last primitive that will be scheduled on TULIP-PE, so that it can be minimized in the objective function

$$\forall v \in V_P, \ \ e_v \leq E. \tag{18}$$

4) Routing Constraints: The following constraints ensure that the data-routing capabilities of local registers are not violated, and are explained as follows. A local register can perform either a read or a write operation at any given time, but not both simultaneously. Therefore, two nodes that share an edge cannot be assigned to the same local register. In Fig. 5, since u is the immediate predecessor of v, the output of u is stored in a different local register than v. While the output of u is read from a local register, the output of v is simultaneously written to a different local register (19). Furthermore, two sibling nodes cannot be assigned the same local register. As shown in Fig. 5, u and v are immediate predecessors of w. Therefore, u and v cannot have the same local register. This constraint is because the local registers supply only one operand to each primitive in TULIP-PE. As a result, we need two separate local registers to provide two operands (20)

$\forall u, v \in V_P \ \forall r \in [0, 3] \ u \prec v$:

$$\rho_{u,r} + \rho_{v,r} \leq 1 \tag{19}$$

$$\rho_{u,r} + \rho_{v,r} + \rho_{w,r} \leq 1. \tag{20}$$

The mapping of the primitive operations to the clusters of TULIP-PE is determined by analyzing the decision variables $\rho_{v,r}$ and s_v. A node v is executed on the cluster associated with the local register r if $\rho_{v,r} = 1$, at the time specified by s_v. The stored data of v is then maintained in the local register till time instance e_v. Table II shows the number of decision variables generated and the time required when using the ILP to generate the schedule of compute graphs of neurons that compute $\sum_{i=0}^{N-1} w_i x_i$ for a varying number of inputs (N). This is a one-time cost to obtain the schedule for QNN nodes on a TULIP-PE. The size of the largest neuron is $N = 4096$ in AlexNet [41].

The ILP described above enables TULIP-PE to modify its schedule depending on the number of neurons enabled in each cluster. Fig. 6(a) and (b) shows how the schedule of addition operation can be varied based on the available neurons (denoted by K). For example, assume that we need to execute an addition operation of two 4-bit numbers, X and Y. TULIP-PE uses five cycles (4 cycles before the next primitive can be launched) to finish its addition operation if only one neuron ($K = 1$) is enabled in each cluster. However, if the number of neurons in each cluster is doubled ($K = 2$), the schedule can be readjusted to finish the addition operation in three cycles (2 cycles before the next primitive can be launched). If all five neurons are enabled in each cluster, then TULIP-PE would only require two cycles (1 cycle before the next primitive can be launched) to finish the addition operation. This critical feature enables a run-time tradeoff between delay and energy efficiency on the TULIP-PE. Furthermore, if some neurons in the manufactured chip stop working, those neurons can be bypassed by modifying the schedule.

VI. MAPPING COMPLETE QNN ON TULIP

In the previous section, we described how a single QNN node, which is a DAG of operations, is executed on a single TULIP-PE. We now present the final step of mapping QNN nodes to TULIP array, taking into account its specific structure, which is shown in Fig. 7. Although the QNN is a DAG, its nodes are arranged in layers with all nodes in a layer performing the same function but on different inputs. As the computations have to proceed layer by layer, the main goal

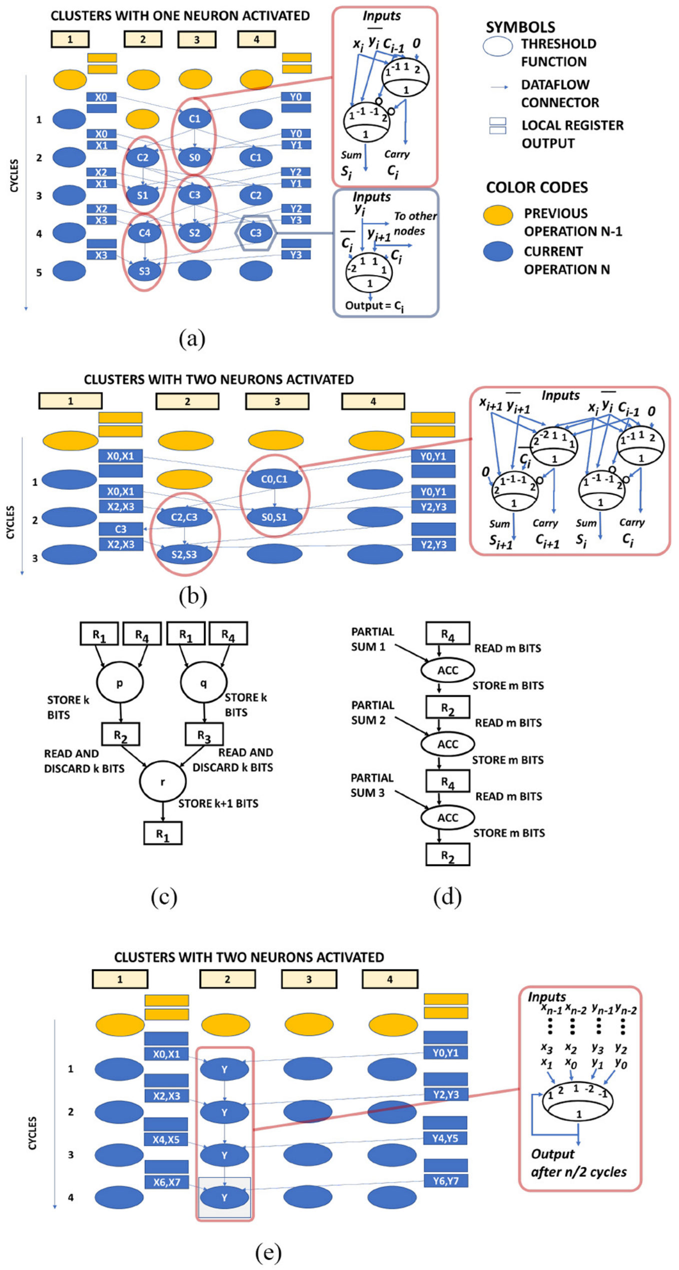


Fig. 6. Addition operation, adder-tree, accumulation, and comparison using the TULIP-PE architecture. Depending on the number of neurons available in each cluster, the scheduler can automatically tune the schedule for the best performance. (a) Addition operation (1-bit per cycle). (b) Addition operation (2-bit per cycle). (c) Addition-tree memory management. (d) Accumulation operation to add partial sums. (e) Comparison operation.

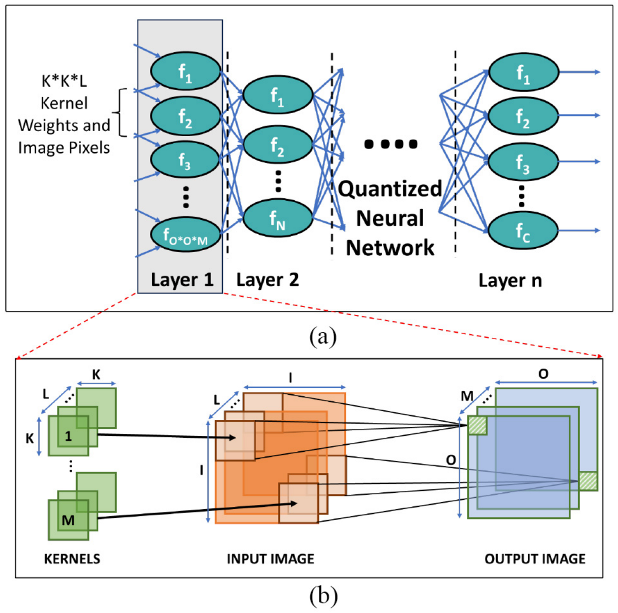


Fig. 7. Representation of QNN as a DAG and data reuse opportunities in 2-D Convolution. (a) QNN as a DAG. (b) Convolution operation.

of improving energy efficiency and latency is achieved by maximizing the data reuse.

Consider a single layer of a QNN that performs a 2-D convolution, as illustrated in Fig. 7(b). The dimensions of the input image are (I, I, L), the output image are (O, O, M) and weights are (K, K, L, M). Each pixel in the output image represents a node in a QNN. For this convolution, the opportunities for data reuse (sharing) are as follows.

1) Each input pixel can be reused $\lfloor K^2O^2/I^2 \rfloor$ times when computing one dimension of the output image.
2) Each kernel weight is reused O^2 times.
3) Each dimension of the input (L) is reused M times.

However, since the data and computation resources required for an arbitrary layer of a QNN might exceed what is available on the TULIP architecture, the nodes of a QNN must be scheduled so that the cost of refetching inputs and weights to the cache from off-chip memory is minimized, subject to the following constraints.

1) A 2-D array of TULIP-PEs operating in an SIMD fashion, such that the TULIP-PEs in the same row share input pixels, and TULIP-PEs in the same column share weights.
2) A fixed cache size for storing input pixels.
3) A fixed cache size for storing weights.

Similar problems have been addressed in several prior works, targeting different platforms [9], [14], [42], [43]. These works minimize data fetches from the external memory by exploiting the fact that the core computations in all CNNs are convolutions and are expressed as deeply nested loops, which can be unrolled either in software or in hardware. The data fetching scheme described in [14] was for a 1-D array of PEs. In this article, we extend the data flow and the node scheduling algorithm presented in [14] for TULIP in a way that maximizes the reuse of input pixels and weights, and achieves high energy efficiency. This allows us to keep the external memory interface uniform between the architectures and allows for a fair comparison between the two architectures.

An illustration of the schedule for the convolution layer of a QNN based on [14] is shown in Fig. 8. Given an image and kernel buffers of a given capacity, a subset of the required data (image pixels and weights) for a convolution operation is loaded from external memory. The computation on the TULIP-PEs is started as soon as the required data is available and partial results are computed. To complete the convolution operation across all input channels (L) and output channels

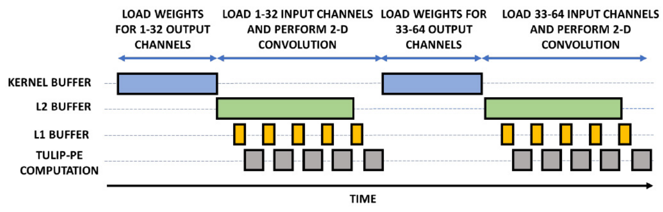

LOAD WEIGHTS FOR 1-32 OUTPUT CHANNELS
LOAD 1-32 INPUT CHANNELS AND PERFORM 2-D CONVOLUTION
LOAD WEIGHTS FOR 33-64 OUTPUT CHANNELS
LOAD 33-64 INPUT CHANNELS AND PERFORM 2-D CONVOLUTION
KERNEL BUFFER
L2 BUFFER
L1 BUFFER
TULIP-PE COMPUTATION
TIME

Fig. 8. Convolution schedule used by TULIP on the basis of algorithms presented in [14].

(M), new input pixels and kernels are loaded to the respective on-chip buffers replacing the previous data.

The architecture presented in [14] has one row with C MAC units (columns) whereas, TULIP has R rows with C TULIP-PEs (columns) which share image pixels along the row and weights along the column. Therefore, at any given instance TULIP computes the convolution operation for R times more output pixels than in [14], with R times higher kernel reuse than [14]. Therefore, the number of external data transfers to load the kernel weights to the kernel buffer reduces by a factor of R in the case of TULIP as compared to [14]. In Section VIII, we show that these comparisons are based on both designs with the same area.

VII. Enhancing Data-Reuse UsingTulip-PEs

This section provides a quantitative analysis to show how the use of TULIP-PEs enhances data reuse, as compared to a MAC unit. This is done by comparing the delay and area complexity when using TULIP-PEs and when using MACs. Let m and n be the number of bits needed to represent inputs and weights, respectively.

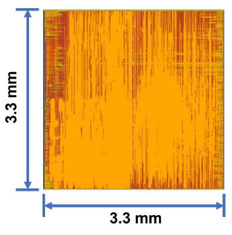
To multiply N pairs of weights and inputs, the area complexity of the MAC unit [44] is $\mathcal{O}(mn)$, whereas for a TULIP-PE it is $\mathcal{O}(1)$. The area complexity of TULIP-PE is a constant because it performs multiplication sequentially, in a bit-sliced manner.

The delay complexity of the MAC unit [44] $\mathcal{O}(N)$ and that of the TULIP-PE is $\mathcal{O}(mnN)$. Although the TULIP-PE is smaller, it is much slower than a MAC unit. However, as explained next, these tradeoffs change when MACs and TULIP-PEs are used in an SIMD architecture.

Consider the following two SIMD architectures. First is the baseline architecture for reference, which consists of a row of C MAC units. Second is the TULIP architecture, with a grid of $R \times C$ grid of TULIP-PEs. The baseline has a gate complexity of $\mathcal{O}(Cmn)$ and a delay complexity of $\mathcal{O}(N/C)$. Similarly, TULIP has a gate complexity of $\mathcal{O}(CR)$ and a delay complexity of $\mathcal{O}(mnN/CR)$. TULIP can match the area and delay of the baseline by setting $R = mn$. However, TULIP is still better than the baseline because the grid arrangement provides higher opportunities for weight reuse. If we assume that a workload of $R \times C$ graphs will be processed by both the architectures, then the baseline would fetch each weight R times whereas the TULIP would fetch each weight just once. As a result, significant energy-efficiency improvements are observed by enhancing data reuse. The complexity analysis discussed above is summarized in Table III.

TABLE III
Gate and Delay Complexity of MAC Units and Tulip-PEs. Tulip-PEs Match the Delay and Gate Complexity of MAC Units When $R = mn$. However, Since There Are Now R Tulip-PEs for Every MAC Unit, the Increased Parallelism Promotes Data Sharing, Thereby Improving Data Reuse by a Factor of R

	Gate complexity	Delay complexity
1 MAC Unit	$\mathcal{O}(mn)$	$\mathcal{O}(N)$
1 TULIP-PE	$\mathcal{O}(1)$	$\mathcal{O}(mnN)$
Row of C MAC units	$\mathcal{O}(Cmn)$	$\mathcal{O}(N/C)$
Grid of $R \times C$ TULIP-PEs	$\mathcal{O}(CR)$	$\mathcal{O}(mnN/CR)$



	TULIP	YodaNN^{++}
Technology	TSMC 40LP	TSMC 40LP
Area (mm^2)	10.6	10.9
L2/L1/Kernel Area (μm^2)	603K/125K /1682K	603K/125K /1682K
Processing Unit Area (μm^2)	1986K	2151K
# Std. Cells	2958K	3258K
# Nets	1548K	1312K
Wirelength (m)	88.67	64.65

Fig. 9. Layout of TULIP architecture in TSMC 40nm-LP.

Note that the concept discussed above has already been used in other design settings. For instance, processor designers often choose to use several slower cores instead of using fewer faster cores, to enhance the energy efficiency without compromising on throughput. The work presented in this article also uses the same concept but at the level of PEs. TULIP replaces the traditionally used MAC units with slower but more energy-efficient TULIP-PEs.

VIII. Experimental Results

A. Experimental Setup

TULIP architecture was evaluated using TSMC 40nm-LP library. Synthesis was done using Cadence Genus, and then the design was placed and routed using Cadence Innovus (Fig. 9). Timing checks were performed using cross-corner analysis at {SS, 125C, 0.81V}, {TT, 25C, 0.9V}, and {FF, 0C, 0.99V}. The VCD file generated using real QNN workloads was used for accurate power analysis by modeling switching activity.

The primitive component of a TULIP-PE is the binary neuron shown in Fig. 1(c). A detailed analysis and design of this cell, along with its advantages over its CMOS functional equivalents appears in [34]. For instance, a 5-input binary neuron in 40nm is about the size of a high-drive strength D-flipflop, but it can replace numerous functions that would normally require several levels of logic implemented using conventional CMOS logic. Overall, at the individual cell level [34] shows that a 5-input binary neuron in 40nm results in improvements in area, power, and delay of [80%, 60%, 40%], respectively, over the performance optimized, functionally equivalent CMOS circuit. These reductions at the individual cell level lead to significant improvements in throughput and energy of the TULIP-PE and of the TULIP architecture.

The three closest comparison points for TULIP are YodaNN [14], UNPU [19], and BitBlade [20]. The UNPU

TABLE IV
COMPARISON OF FULLY RECONFIGURABLE MAC UNIT BASED ON THE YODANN ARCHITECTURE [14], WITH A TULIP-PE (K=5), FOR COMPUTING A 288 INPUT WEIGHTED SUM (32 INPUT CHANNELS, KERNEL =3×3). TULIP-PE IS 15.8× SMALLER THAN THE MAC UNIT. PDP: POWER DELAY PRODUCT

	Bit width	1	2	3	4
TULIP-PE (4366 μm^2)	Power (mW)	0.18	0.18	0.18	0.18
	Cycles (#)	155	170	227	307
	Time (ns)	356.5	391.0	522.1	706.1
	PDP (pJ)	65.5	71.8	95.9	129.7
MAC Unit (69028 μm^2)	Power (mW)	2.6	8.0	16.2	22.6
	Cycles (#)	32	32	32	32
	Time (ns)	73.6	73.6	73.6	73.6
	PDP (pJ)	189.4	590.3	1193.8	1666.3
Ratio (X=B/T)	PDP	2.9	8.2	12.4	12.8

architecture contains a full processor core, memory controller, etc., and hence it is harder to reproduce. Simillarly, the BitBlade architecture also uses a CPU for some operations of the QNN. Furthermore, the data presented in this article is all relative to another benchmark architecture. The energy numbers and throughput numbers are normalized to their chosen benchmark. Thus, it is difficult to perform a meaningful quantitative comparison with UNPU and BitBlade. On the other hand, YodaNN paper had sufficient details that allowed us to reproduce the architecture reliably. Note that both architectures are similar fundamentally since they use accumulator-based PEs. Since the main focus of this article is to highlight how TULIP-PE avoids hardware overprovisioning while also supporting a variety of QNN operations, the comparison was done against the YodaNN architecture. YodaNN is a BNN accelerator that was designed in 65nm technology. To present a fair comparison, we implemented the complete design of YodaNN using TSMC 40nm-LP technology and extended the design to support 2 to 4 bit QNNs. Our implementation of YodaNN will be referred to as YodaNN^{++}.

Although YodaNN [14] does not report the throughput and energy efficiency for fully connected layers, we estimated them by performing element-wise matrix multiplications using the MAC units present in their architecture. In summary, TULIP and YodaNN^{++} were both designed in the same technology, with the same memory organization, with support for 12-bit inputs, support for up to 4-bit weights and activations and kernel sizes of 3, 5, and 7.

B. *Evaluation of* TULIP-*PE Against MAC*

Table IV compares the baseline 18-bit reconfigurable MAC unit used in YodaNN^{++} with a TULIP-PE with five neurons in each cluster and a 16-bit local register for each neuron. In large QNN architectures such as Alexnet [45], the input layers

TABLE V
ENERGY EFFICIENCY [EN. EFF. (TOP/J)] AND THROUGHPUT (GOP/S) OF TULIP AND YODANN^{++} FOR CIFAR-10 CLASSIFICATION. K INDICATES THE NUMBER OF NEURONS USED IN EACH CLUSTER. TWO VARIANTS OF TULIP ARE SHOWN: ONE IS TUNED FOR ENERGY EFFICIENCY, WHILE THE OTHER IS TUNED FOR PERFORMANCE. (A) ALEXNET. (B) RESNET18. (C) RESNET20

I/W bits	YodaNN^{++}		TULIP-Q for En. Eff.			TULIP-Q for Perf		
	En. Eff.	Perf.	En. Eff.	Perf.	K	En. Eff.	Perf.	K
(a)								
1	3.1	66.6	142.4 (46.7X)	86.9 (1.3X)	1	86.2 (28.3X)	92.7 (1.4X)	5
2	1.0	66.6	43.8 (44.3X)	79.7 (1.2X)	2	30.3 (30.6X)	86.7 (1.3X)	5
3	0.5	66.6	23.3 (47.6X)	71.6 (1.1X)	3	18.3 (37.4X)	78.9 (1.2X)	5
4	0.4	66.6	14.9 (42.5X)	64.3 (1.0X)	4	13.1 (37.4X)	69.4 (1.0X)	5
(b)								
1	2.4	31.6	108.6 (44.9X)	39.7 (1.3X)	1	65.8 (27.2X)	44.3 (1.4X)	5
2	0.8	31.6	32.2 (40.7X)	34.8 (1.1X)	2	22.8 (28.8X)	40.0 (1.3X)	5
3	0.4	31.6	17.0 (43.6X)	31.0 (1.0X)	3	13.7 (35.2X)	36.1 (1.1X)	5
4	0.3	31.6	10.9 (38.8X)	28.7 (0.9X)	4	9.8 (34.9X)	32.5 (1.0X)	5
(c)								
1	2.3	53.3	135.7 (59.5X)	90.8 (1.7X)	1	85.8 (37.6X)	117.3 (2.2X)	5
2	0.7	53.3	40.9 (55.2X)	66.9 (1.3X)	2	29.9 (40.3X)	86.2 (1.6X)	5
3	0.4	53.3	21.8 (60.4X)	52.4 (1.0X)	3	18.0 (50.0X)	66.2 (1.2X)	5
4	0.3	53.3	14.0 (53.7X)	44.3 (0.8X)	4	12.8 (49.3X)	52.7 (1.0X)	5

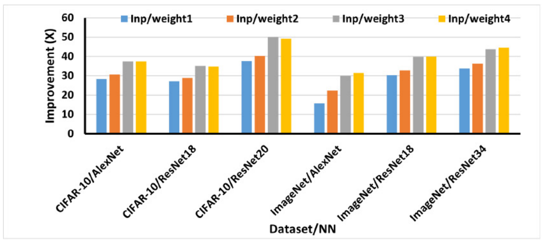
are integer, while the other layers are quantized. Consequently, both the MAC unit and TULIP-PE can support both types of layers. The MAC unit and TULIP-PE are compared when computing the outputs of the quantized layers. Both modules perform the weighted sum with quantized activations and weights. The MAC unit realizes convolution by multiplying and accumulating one kernel window in each cycle. On the other hand, the TULIP-PE treats convolution as a weighted sum represented as a compute graph of multiplication operations connected to an adder tree. This is important because TULIP realizes adders, multipliers, etc., of custom bit widths, thereby reducing the energy incurred by MAC unit that uses maximum width addition and multiplication operations in every cycle.

Table IV shows that the TULIP-PE is 15.8× smaller than the MAC unit and consumes up to 125× less power. However, its delay is 9.5× higher than the MAC unit since it performs bit-level addition. As a result, the power delay product of a TULIP-PE is up to 5.8× lower than the MAC unit while at the same time being 15.8× smaller than the MAC. Furthermore, since a MAC unit cannot compute operations, such as comparison, max-pooling, etc., the data is sent to other parts of the chip for these operations in the baseline [14]. However, the TULIP-PE preserves the data locality and performs the comparison and max-pooling operations using the same hardware, without moving the data to other modules, thus resulting in additional energy savings. The reduced area allows us to have more TULIP-PEs, which leads to higher throughput.

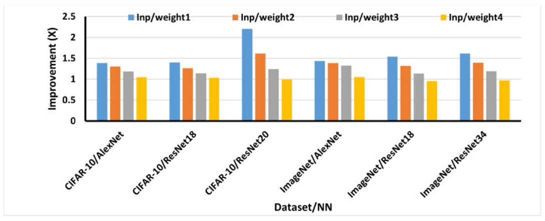
C. *Evaluation of the* TULIP *Architecture*

The implementation of TULIP has 512 TULIP-PEs. This is to ensure that the area of YodaNN^{++} and that of TULIP are the same. Note that the number of processing units in TULIP can easily be scaled to suit the application. For both designs, the size of the L1 buffer is 2.3 kB, the size of the L2 buffer is 10.5 kB, and the size of the kernel buffer is 24.5 kB.

Tables V and VI show the energy efficiency and throughput values for various neural networks (at varying bit-precisions), accelerated using both the TULIP and YodaNN^{++}. For TULIP,

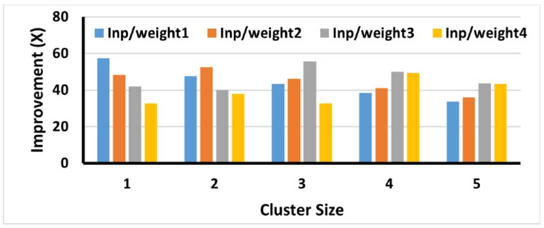


(a)

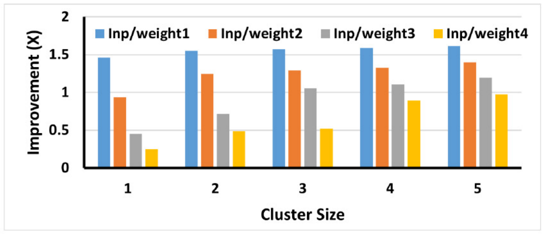


(b)

Fig. 10. Improvements of TULIP (using five neurons per cluster) over YodaNN^{++}, for various neural networks. (a) Improvements in energy efficiency. (b) Improvements in throughput.



(a)



(b)

Fig. 11. Improvements of TULIP (with varying number of active neurons in each cluster) over YodaNN^{++}, for ImageNet Classification using ResNet-34. (a) Improvements in energy efficiency. (b) Improvements in throughput.

TABLE VI
ENERGY EFFICIENCY [EN. EFF. (TOP/J)] AND THROUGHPUT (GOP/S) OF TULIP AND YODANN^{++} FOR IMAGENET CLASSIFICATION. K INDICATES THE NUMBER OF NEURONS USED IN EACH CLUSTER. TWO VARIANTS OF TULIP ARE SHOWN: ONE IS TUNED FOR ENERGY EFFICIENCY, WHILE THE OTHER IS TUNED FOR PERFORMANCE . (A) ALEXNET. (B) RESNET18. (C) RESNET34

I/W	YodaNN^{++}		TULIP-Q for En. Eff.			TULIP-Q for Perf		
bits	En. Eff.	Perf.	En. Eff.	Perf.	K	En. Eff.	Perf.	K
(a)								
1	4.3	123.2	84.2 (19.8X)	172.8 (1.4X)	4	66.5 (15.7 X)	177.2 (1.4X)	5
2	1.7	123.2	38.5 (22.4X)	171.4 (1.4X)	5	38.5 (22.4 X)	171.4 (1.4X)	5
3	0.9	123.2	27.0 (30.0X)	163.4 (1.3X)	5	27.0 (30.0 X)	163.4 (1.3X)	5
4	0.7	123.2	20.8 (31.5X)	129.3 (1.0X)	5	20.8 (31.5 X)	129.3 (1.0X)	5
(b)								
1	3.2	84.6	139.3 (43.9X)	111.7 (1.3X)	1	95.8 (30.2X)	130.7 (1.5X)	5
2	1.0	84.6	43.8 (42.1X)	92.7 (1.1X)	2	34.2 (32.9X)	111.9 (1.3X)	5
3	0.5	84.6	23.8 (45.8X)	78.9 (0.9X)	3	20.7 (39.8X)	96.0 (1.1X)	5
4	0.4	84.6	15.6 (42.2X)	69.4 (0.8X)	4	14.8 (40.0X)	80.9 (1.0X)	5
(c)								
1	3.1	88.2	178.0 (57.4X)	129.0 (1.5X)	1	104.9 (33.8X)	142.5 (1.6X)	5
2	1.0	88.2	52.9 (52.9X)	109.8 (1.2X)	2	36.3 (36.3X)	123.1 (1.4X)	5
3	0.5	88.2	27.9 (55.8X)	93.1 (1.1X)	3	21.9 (43.7X)	105.2 (1.2X)	5
4	0.4	88.2	17.7 (50.5X)	78.6 (0.9X)	4	15.6 (44.5X)	86.0 (1.0X)	5

two sets of results are presented: 1) TULIP tuned for the best energy efficiency and 2) TULIP tuned for the best throughput. Here, tuning is done by changing the number of active neurons (cluster size K) in each cluster. Based on Tables V and VI, TULIP shows consistent improvement in the energy efficiency over YodaNN^{++} for all the neural networks.

Fig. 10(a) shows that TULIP consistently achieves an order of magnitude improvement in energy efficiency for all variants of the neural networks. This is primarily attributed to the fact that TULIP realizes adders, multipliers, etc., of different bit widths, which eliminates the waste incurred by conventional accumulation methods that use operators to accommodate the maximum width. This, coupled with the improved weight reuse, results in a substantial improvement in energy efficiency over YodaNN^{++}.

Fig. 10(b) shows throughput can be improved by reducing the bit precision. This is because fewer bits need to be processed for each operation. Fig. 11(b) shows that the throughput increases as the number of neurons in each cluster increases. Although this graph is restricted to the inference of Imagenet classification using ResNet-34, this trend applies to other neural networks as well. The increase in the throughput due to the increase in the number of neurons allows each operation to execute faster on the TULIP-PE. By appropriately choosing the right configuration, it is possible to match the throughput of the baseline (or even improve it) while gaining significant improvements in energy efficiency. The corresponding improvements in energy efficiency are shown in Fig. 11(a).

Fig. 12 demonstrates how the TULIP architecture can be used to tradeoff energy efficiency and accuracy at runtime for neural networks used for ImageNet classification tasks. As the bit-precision increases, the energy efficiency decreases but accuracy increases. Hence, accuracy can be traded off at run-time with energy efficiency and throughput. This would be particularly useful for energy-constrained mobile devices where high accuracy may not be necessary to make the correct decision, or conversely, the accuracy could suddenly be increased in a critical situation, after operating at a lower precision.

In summary, there are two key observations that can be made from the experimental results.

1) TULIP can support multiple bit-precision of the weights and activations of a QNN and achieves consistently greater energy efficiency than the baseline architecture for the same.
2) TULIP enables a high degree of *tunability* at run-time, to tradeoff energy efficiency, throughput, and accuracy.

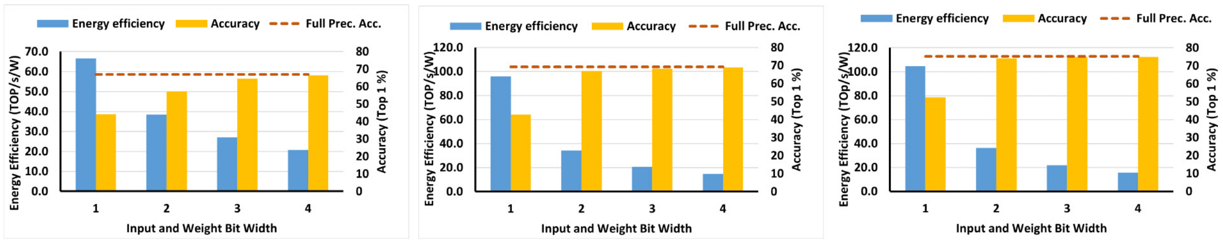


Fig. 12. Trading off energy efficiency [En. Eff (TOP/J)] with accuracy (%) for ImageNet Classification using the TULIP architecture. Full Prec. Acc. indicates top the 1% accuracy when using 32-bit integers and weights. (a) ImageNet/AlexNet. (b) ImageNet/ResNet-18. (c) ImageNet/ResNet-34.

IX. CONCLUSION

This article presents a new design of a QNN accelerator, called TULIP, that uses binary neurons as its core compute elements. TULIP and the baseline design YodaNN^{++} were designed to the layout level and simulated on several well known neural networks. The simulations were carried out using commercial libraries and design tools and account for all the device, circuit and layout characteristics. The results show that TULIP can improve the energy efficiency by 30×–50× when compared the baseline design YodaNN^{++}, with both designs having approximately the same area. These improvements do not rely on standard low-power techniques such as voltage scaling and approximate computing. The improvements in energy efficiency can be attributed to several factors: 1) the use of operators of the required bit-width instead of the maximum bit width; 2) the use of artificial neurons to compute complex logic functions within a very small area, thereby allowing for greater number of PEs for parallel operations; and 3) the ability to reconfigure the function of the neurons without sacrificing performance or energy efficiency. TULIP allows for tuning the precision and throughput and energy efficiency.

APPENDIX

A. Problem of Scheduling QNN Node on TULIP-PE

In this section, we provide a precise formulation of the problem of *register-aware* scheduling of a QNN node on a TULIP-PE. It finds a mapping between two graphs: 1) a DFG of threshold functions that represents a QNN node and 2) a *time-extended resource graph* that represents TULIP-PE.

Definition 2 (Threshold Graph Gth(Vth, Eth)): This is a DAG where the nodes Vth represent threshold functions and an edge $(u, v) \in E$th means that the output of u is an input of the threshold function v.

Definition 3 (Time Extended Resource Graph (TERG) $G_R(V_R, E_R)$): This is a DAG where a node is a pair $(u, t) \in V_R$, u is a local register or a neuron and t is a time instance at which the resource u is available. An edge between two resources (u, t) and $(v, t+1)$ is represented as (u, v, t) to indicate that the output of u is input to v at time $t+1$. Edges are absent between resources if their timestamps differ by more than 1 or if there is no physical datapath between their associated neurons (or local registers). The *latency* of G_R is $\max_{(u,t)\in V_R}\{t\}$.

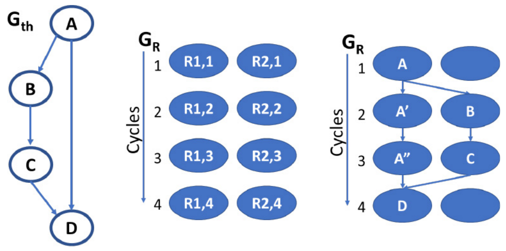


Fig. 13. Scheduling graphs of threshold functions on binary neurons. (a) Compute graph G_{th}. (b) Time-extended resource graph G_R. (c) Mapping solution of G_{th} to G_R.

Definition 4 (Feasible Schedule): Let $V_R^* \subseteq V_R$, and let $M : V_R^* \to V_{th}$ denote a surjective mapping, i.e., $M(V_R^*) = V_{th}$. For definiteness, let $M = \{(v_1, t_1) \to u_1, (v_2, t_2) \to u_2), \ldots, (v_n, t_n) \to u_n\}$.

A feasible schedule is a pair $(G_R(V_R^*, E_R), M : V_R^* \to V_{th})$, where G_R is TERG on V_R^* and for each $(u, v) \in V_{th}$, there exists a path $P = \{(r_0, t_0), \ldots, (r_k, t_k), (r_{k+1}, t_{k+1})\}$ in G_R such that $M(r_0) = u$, $M(r_{k+1}) = v$, and $t_{i+1} = t_i + 1$, $0 \le i \le k$, and $k \ge 0$.

Fig. 13 shows an example mapping of a given compute graph containing four nodes to a time-extended resource graph containing two resources, extended over four cycles.

Definition 5 (TGS Problem): Given a threshold graph $G_{th}(V_{th}, E_{th})$, the TGS problem is to construct a feasible schedule of minimum latency.

In the design of the TULIP-PE shown in Fig. 1(b), there are four clusters, each with five neurons and each with a 16-bit register. Consequently, the number of resources will be 320, for each timestamp t. The maximum number of timesteps in the extended resource graph would be the number of levels in the compute graph.

The formulation of the TGS problem presented above is the same as the well-studied problem of mapping a DFG of computations onto a CGRA. There is an extensive body of literature on CGRA architectures and scheduling computations onto them that spans more than two decades. A precise formulation of the CGRA scheduling problem, similar to the TGS problem, first appeared in [39] and was shown to be NP-complete. This was subsequently extended to register-aware mapping in [46], followed by several extensions [47], [48], [49], [50], [51].

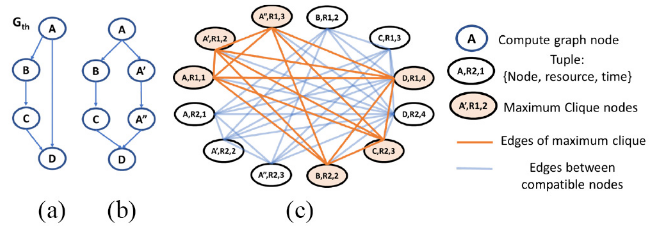


Fig. 14. Scheduling threshold function graphs on binary neurons. (a) Compute graph G_{th}. (b) Equivalent transformed graph G'_{th}. (A' and A'' indicate buffer functions). (c) Compatibility graph of G'_{th} and G_R .

Mapping the computations within the innermost loop involves small computation graphs, in the order of tens of nodes. In addition, with the target architecture being a CGRA, the resulting resource graph is also of the same order. This allows the various heuristic algorithms for finding a feasible mapping to enumerate the possibilities by transforming the compute graph [Fig. 14(b)], constructing a *compatibility graph* [Fig. 14(c)] and then finding a maximal clique of that graph, at each time step. The size of the compatibility graph is the product of the sizes of the computation and resource graphs. Such an approach is not possible for the TGS problem because the size of the compatibility graph would be in the tens of thousands for which a maximal clique has to be computed. For this reason, an alternate approach is presented in Section V.

Note that traditional high-level scheduling algorithms (used for task allocation to CPUs [40], [52]) do not apply when performing scheduling on TULIP-PE. This is because, unlike high-level scheduling algorithms, routing-aware scheduling algorithms used for CGRAs generate a valid schedule while also honoring the routing constraints that arise due to the physical limitations (bandwidth, connectivity, etc.) of the hardware.

REFERENCES

[1] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *J. Mach. Learn. Res.*, vol. 23, no. 120, pp. 1–39, 2022.
[2] S. Kim, J. Lee, S. Kang, J. Lee, and H.-J. Yoo, "A power-efficient CNN accelerator with similar feature skipping for face recognition in mobile devices," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 4, pp. 1181–1193, Apr. 2020.
[3] Z. Liu, P. N. Whatmough, Y. Zhu, and M. Mattina, "S2TA: Exploiting structured sparsity for energy-efficient mobile CNN acceleration," in *Proc. IEEE Int. Symp. High Perform. Comput. Architect. (HPCA)*. Los Alamitos, CA, USA, 2022, pp. 573–586.
[4] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC," in *Proc., Int. Conf. Field Program. Technol. (FPT)*, 2016, pp. 77–84.
[5] A. Boutros, B. Grady, M. Abbas, and P. Chow, "Build fast, trade fast: FPGA-based high-frequency trading using high-level synthesis," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs (ReConFig)*, 2017, pp. 1–6.
[6] Y. Ma, Y. Cao, S. Vrudhula, N. Suda, and J. Sun Seo, "ALAMO: FPGA acceleration of deep learning algorithms with a modularized RTL compiler," *Integr., VLSI J.*, vol. 62, pp. 14–23, Jun. 2018.
[7] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing the convolution operation to accelerate deep neural networks on FPGA," *IEEE Trans. VLSI*, vol. 26, no. 7, pp. 1354–1367, Jul. 2018.
[8] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Automatic compilation of diverse CNNs onto high-performance FPGA accelerators," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst. TCAD*, vol. 39, no. 2, pp. 424–437, Feb. 2020.
[9] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Performance modeling for CNN inference accelerators on FPGA," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst. TCAD*, vol. 39, no. 4, pp. 843–856, Apr. 2020.
[10] H.-S. Suh, J. Meng, T. Nguyen, V. Kumar, Y. Cao, and J.-S. Seo, "Algorithm-hardware co-optimization for energy-efficient drone detection on resource-constrained FPGA," *ACM Trans. Reconfig. Technol. Syst.*, vol. 16, no. 2, pp. 1–25, May 2023.
[11] X. Sun, X. Peng, P. Chen, R. Liu, J. Seo, and S. Yu, "Fully parallel RRAM synaptic array for implementing binary neural network with (+1, −1) weights and (+1, 0) neurons," in *Proc. 23rd Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2018, pp. 574–579.
[12] D. Fan and S. Angizi, "Energy efficient in-memory binary deep neural network accelerator with dual-mode SOT-MRAM," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, 2017, pp. 609–612.
[13] X. Guo et al., "Fast, energy-efficient, robust, and reproducible mixed-signal neuromorphic classifier based on embedded NOR flash memory technology," in *Proc. IEEE Int. Electron Devices Meet. (IEDM)*, 2017, pp. 6.5.1–6.5.4.
[14] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An architecture for ultralow power binary-weight CNN acceleration," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 48–60, Jan. 2018.
[15] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.
[16] A. Al Bahou, G. Karunaratne, R. Andri, L. Cavigelli, and L. Benini, "XNORBIN: A 95 TOp/s/W hardware accelerator for binary convolutional neural networks," in *Proc. IEEE Symp. Low Power High Speed Chips*, 2018, pp. 1–3.
[17] P. C. Knag et al., "A 617 TOPS/W all digital binary neural network accelerator in 10nm FinFET CMOS," in *Proc. IEEE Symp. VLSI Circuits*, 2020, pp. 1–2.
[18] B. Moons, D. Bankman, L. Yang, B. Murmann, and M. Verhelst, "BinarEye: An always-on energy-accuracy-scalable binary CNN processor with all memory on chip in 28nm CMOS," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, 2018, pp. 1–4.
[19] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE J. Solid State Circuits*, vol. 54, no. 1, pp. 173–185, Jan. 2019.
[20] S. Ryu et al., "BitBlade: Energy-efficient variable bit-precision hardware accelerator for quantized neural networks," *IEEE J. Solid State Circuits*, vol. 57, no. 6, pp. 1924–1935, Jun. 2022.
[21] Z. Wang, M. Agung, R. Egawa, R. Suda, and H. Takizawa, "Automatic hyperparameter tuning of machine learning models under time constraints," in *Proc. IEEE Int. Conf. Big Data*, 2018, pp. 4967–4973.
[22] Y. Wang, Y. Wang, H. Li, Z. Cai, X. Tang, and Y. Yang, "CNN hyperparameter optimization based on CNN visualization and perception hash algorithm," in *Proc. 19th Int. Symp. Distrib. Comput. Appl. Bus. Eng. Sci. (DCABES)*, 2020, pp. 78–82.
[23] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2017, pp. 5068–5076.
[24] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 6071–6079.
[25] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 4820–4828.
[26] Y. Wang, H. Shen, and D. Duan, "On stabilization of quantized sampled-data neural-network-based control systems," *IEEE Trans. Cybern.*, vol. 47, no. 10, pp. 3124–3135, Oct. 2017.
[27] J. Yue et al., "14.3 a 65nm computing-in-memory-based CNN processor with 2.9-to-35.8TOPS/W system energy efficiency using dynamic-sparsity performance-scaling architecture and energy-efficient inter/intra-macro data reuse," in *Proc. IEEE Int. Solid State Circuits Conf. (ISSCC)*, 2020, pp. 234–236.
[28] C. Luo, J. Diao, and C. Chen, "FullReuse: A novel ReRAM-based CNN accelerator reusing data in multiple levels," in *Proc. IEEE 5th Int. Conf. Integr. Circuits Microsyst. (ICICM)*, 2020, pp. 177–183.
[29] E. Ahanonu, M. Marcellin, and A. Bilgin, "Lossless image compression using reversible integer wavelet transforms and convolutional neural networks," in *Proc. Data Compress. Conf.*, 2018, p. 395.
[30] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 126–136, Jan. 2018.

[31] A. Trusov, E. Limonova, D. Slugin, D. Nikolaev, and V. V. Arlazarov, "Fast implementation of 4-bit convolutional neural networks for mobile devices," in *Proc. 25th Int. Conf. Pattern Recognit. (ICPR)*, 2020, pp. 9897–9903.

[32] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," 2021, *arXiv:2106.08295*.

[33] H. Nakahara, H. Yonekawa, T. Sasao, H. Iwamoto, and M. Motomura, "A memory-based realization of a binarized deep convolutional neural network," in *Proc. Int. Conf. Field Program. Technol. (FPT)*, 2016, pp. 277–280.

[34] A. Wagle, G. Singh, S. Khatri, and S. Vrudhula, "A novel ASIC design flow using weight-tunable binary neurons as standard cells," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 7, pp. 2968–2981, Jul. 2022.

[35] A. Wagle, S. Khatri, and S. Vrudhula, "A configurable BNN ASIC using a network of programmable threshold logic standard cells," in *Proc. IEEE 38th Int. Conf. Comput. Design (ICCD)*, 2020, pp. 433–440.

[36] S. Muroga, *Threshold Logic and Its Applications*. New York, NY, USA: Wiley, Inc., 1971.

[37] J. Yang, N. Kulkarni, S. Yu, and S. Vrudhula, "Integration of threshold logic gate circuit with RROM devices for low power, and robust operation," in *Proc. IEEE/ACM Int. Symp. Nanoscale Archit.*, Paris, France, 2014, pp. 39–44.

[38] N. Kulkarni, J. Yang, J.-S. Seo, and S. Vrudhula, "Reducing power, leakage and area of standard cell ASICs using threshold logic flipflops," *IEEE Trans. VLSI*, vol. 24, no. 9, pp. 2873–2886, Sep. 2016.

[39] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "EPIMap: Using epimorphism to map applications on CGRAs," in *Proc. 49th Design Autom. Conf. (DAC)*, San Diego, CA, USA, 2012, pp. 1280–1287.

[40] B. Landwehr, P. Marwedel, and R. Dömer, *OSCAR: Optimum Simultaneous Scheduling, Allocation and Resource Binding Based on Integer Programming*, Univ. Dortmund, Dortmund, Germany, Sep. 1994, pp. 90–95.

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Conf. Neural Inf. Process. Syst. (NIPS)*, 2012, pp. 1–9.

[42] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid State Circuits (JSSC)*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[43] S. Dave, Y. Kim, S. Avancha, K. Lee, and A. Shrivastava, "DMazeRunner: Executing perfectly nested loops on dataflow accelerators," *ACM Trans. Embed. Comput. Syst.*, vol. 18, no. 5, pp. 1–27, Oct. 2019.

[44] J. Garland and D. Gregg, "Low complexity multiply–accumulate units for convolutional neural networks with weight-sharing," *ACM Trans. Archit. Code Optim.*, vol. 15, no. 3, pp. 1–24, Sep. 2018.

[45] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2016, pp. 525–542.

[46] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "REGIMap: Register-aware application mapping on coarse-grained reconfigurable architectures (CGRAs)," in *Proc. 50st ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, 2013, pp. 1–10.

[47] M. Hamzeh, A. Shrivastava, and S. Vrudhula, "Branch-aware loop mapping on CGRAs," in *Proc. 51st ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, 2014, pp. 1–6.

[48] Y. Chen, Z. Zhao, J. Jiang, G. He, Z. Mao, and W. Sheng, "Reducing memory access conflicts with loop transformation and data reuse on coarse-grained reconfigurable architecture," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2021, pp. 124–129.

[49] M. Canesche et al., "TRAVERSAL: A fast and adaptive graph-based placement and routing for CGRAs," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, vol. 40, no. 8, pp. 1600–1612, Aug. 2021.

[50] M. Balasubramanian and A. Shrivastava, "CRIMSON: Compute-intensive loop acceleration by randomized iterative modulo scheduling and optimized mapping on CGRAs," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3300–3310, Nov. 2020.

[51] M. Balasubramanian, S. Dave, A. Shrivastava, and R. Jeyapaul, "LASER: A hardware/software approach to accelerate complicated loops on CGRAs," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2018, pp. 1069–1074.

[52] D. Ku and G. De Micheli, "Relative scheduling under timing constraints," in *Proc. 27th ACM/IEEE Design Autom. Conf.*, New York, NY, USA, 1991, pp. 59–64. [Online]. Available: https://doi.org/10.1145/123186.123227

Ankit Wagle (Member, IEEE) received the B.S. degree in electronics and telecommunication from the University of Pune, Pune, India, in 2013, and the M.S. degree in VLSI design from Vellore Institute of Technology, Vellore, India, in 2015. He is currently pursuing the Ph.D. degree with Arizona State University, Tempe, AZ, USA.

He did graduate research internships with Intel, Bengaluru, India, and Maxlinear, Carlsbad, CA, USA, in 2015 and 2017, respectively. He worked with Open-Silicon, Bengaluru, from 2015 to 2016. His research focuses on creating energy-efficient digital circuits using threshold-logic gates and designing neural network accelerators.

Gian Singh (Member, IEEE) received the B.Tech. degree in electronics and communication engineering from the National Institute of Technology Hamirpur (NIT-H), Hamirpur, India, in 2017. He is currently pursuing the Ph.D. degree in computer engineering with Arizona State University, Tempe, AZ, USA.

He has interned with Maxlinear, Carlsbad, CA, USA; Qualcomm, San Jose, CA, USA; and Micron, Boise, ID, USA. He was a Project Associate with NIT-H for the Government of India's SMDP-C2SD Project from 2017 to 2018. His current research focuses on energy-efficient, high-throughput systems for data-intensive applications using artificial neurons and in-memory architectures.

Sunil Khatri (Senior Member, IEEE) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology Kanpur, Kanpur, India, in 1987, the M.S. degree in electronics and communication engineering from The University of Texas at Austin, Austin, TX, USA, in 1989, and the Ph.D. degree in electrical engineering and computer sciences from the University of California at Berkeley, Berkeley, CA, USA, in 1999.

He is currently a Professor of Electronics and Communication Engineering with Texas A&M University at College Station, College Station, TX, USA. He has authored or coauthored more than 250 peer-reviewed publications. Among these papers, five received a best paper award, while six others received best paper nominations. He has coauthored nine research monographs and one edited research monograph, three book chapters, and 13 invited conference papers or workshop papers. He was invited to serve as a Panelist at a conference seven times and have presented two conference tutorials. He holds six U.S. patents. His current research interests include VLSI IC/system-on-a-chip design [including energy-efficient design of custom ICs and field-programmable gate arrays (FPGAs), radiation- and variation-tolerant design, and clocking], algorithm acceleration using hardware (FPGA as well as custom IC based) and software (uniprocessor and GPU based), and interdisciplinary extensions of these topics to other areas.

Sarma Vrudhula (Life Fellow, IEEE) received the B.Math. degree from the University of Waterloo, Waterloo, ON, Canada, in 1976, and the M.S.E.E. and Ph.D. degrees in electrical and computer engineering from the University of Southern California, Los Angeles, CA, USA, in 1980 and 1985, respectively.

He is a Professor with the School of Computing and AI, Arizona State University, Tempe, AZ, USA, and the Director of the NSF I/UCRC for Intelligent, Distributed, Embedded Applications and Systems which was established in 2023. His work spans several areas in design automation and computer-aided design for digital integrated circuit and systems, focusing on low-power circuit design, and energy management of circuits and systems. His specific topics include energy optimization of battery-powered computing systems, including smartphones, wireless sensor networks and IoT systems that rely on energy harvesting; system-level dynamic power and thermal management of multicore processors and system on chip; statistical methods for the analysis of process variations; statistical optimization of performance, power, and leakage; new circuit architectures of threshold-logic circuits for the design of ASICs and FPGAs, and HW accelerators for AI/ML applications.