# uSystolic: Byte-Crawling Unary Systolic Array

Di Wu and Joshua San Miguel
*Department of ECE, University of Wisconsin–Madison, Madison, WI, USA*
di.wu@ece.wisc.edu and jsanmiguel@wisc.edu

***Abstract*—General matrix multiply (GEMM) is an important operation in broad applications, especially the thriving deep neural networks. To achieve low power consumption for GEMM, researchers have already leveraged unary computing, which manipulates bitstreams with extremely simple logic. However, existing unary architectures are not well generalizable to varying GEMM configurations in versatile applications and incompatible to the binary computing stack, imposing challenges to execute unary GEMM effortlessly. In this work, we address the problem by architecting a hybrid unary-binary systolic array, uSystolic, to inherit the legacy-binary data scheduling with slow (thus power-efficient) data movement, i.e., data bytes are crawling out from memory to drive uSystolic. uSystolic exhibits tremendous area and power improvements as a joint effect of 1) low-power computing kernel, 2) spatial-temporal bitstream reuse, and 3) on-chip SRAM elimination. For the evaluated edge computing scenario, compared with the binary parallel design, the rated-coded uSystolic reduces the systolic array area and total on-chip area by 59.0% and 91.3%, with the on-chip energy and power efficiency improved by up to 112.2× and 44.8× for AlexNet.**

## I. Introduction

Prior research has shown that general matrix multiply (GEMM), as the core of deep neural networks (DNNs), dominates over 90% computation of the entire workload [28]. As such, in the past decade, tons of research efforts have been put on optimizing GEMM operations, including both matrix convolution and matrix multiplication, from varying aspects, like energy [10], power [51], latency [64] and throughput [60], etc. Most of those optimizations apply binary computing architectures with parallel bits as data, which have been pervasively adopted in real-world applications for decades. The correspondent binary computing stacks have been well developed for GEMM, from programming languages and compilers [8], [14], [20], [65] to microarchitectures and architectures [30], [41], [66], [77]. Those optimizations are not only applicable to high performance computing in datacenters, but also compatible to rapidly-evolving edge computing with a restricted power budget, e.g., in mobile systems.

***Challenges for Low-Power GEMM Execution.*** Despite the holistic and mature design stack, low-power GEMM execution on edge devices is still challenging, even considering the recent technology advances. First, *binary computing usually equips the computing kernels with on-chip SRAM, increasing power consumption.* Conventional binary GEMM kernels compute within short cycles and require frequent memory accesses, which leads to high power consumption, e.g., remote DRAM accesses occupy around 95% of the total power in [9], as DRAM accesses consume three orders of magnitude more energy than fixed-point adders [19]. To reduce remote DRAM accesses for high efficiency, on-chip SRAM is slabbed on to the computing kernels, like systolic array [30], [34] and 2-D mesh [13] (B-Systolic and B-Mesh in Table I), for local data reuse. Though DRAM power is reduced, SRAM now swallows power. Second, *the power of parallel binary computing kernels increases superquadratically with the data bitwidth.* The gate count and power of binary computing kernels, fed on parallel bits, increase quadratically with the data bitwidth. Moreover, the routing congestion [68], [74] further exacerbates the power overhead in a superquadratical manner and limits the scalability, especially for the 2-D mesh without local interconnections as in systolic array. As such, though precise and highly generalizable for varying DNN applications, i.e., flexibly configuring GEMM to varying shapes (matrix shape) and types (either matrix convolution or multiplication), binary systolic array and mesh architectures are not offering high power efficiency. Alternatively, emerging unary computing utilizes extremely simple logic to operate serial bitstreams, either rate or temporal coding, resulting in power efficiency over binary computing [16], [43], [69]. However, its specific challenge is that *existing low-power unary GEMM architectures are not efficiently generalizable to diverse DNN-based applications on the edge due to their dedication to specific GEMM configurations.* Leveraging the high computation density thanks to the simple logic, prior works mostly exploit fully streaming unary architectures (FSU in Table I) [40], [43], [54], [61], [68], [69], [75], i.e., no intermediate interconversion between binary data and unary bitstreams before the final output [69], to execute DNNs in a fully parallel manner and eliminate the requirement for data scheduling. However, DNN applications on the edge have varying GEMM configurations [5], [18], [19], [23], [25], [26], [50], [76], where one single FSU architecture fails to serve all purposes with high generalizability. As a result, conventional FSU architectures force the coexistence of multiple unary hardware, eventually diminishing the area and power benefits. Similar to binary computing, FSU architectures for large GEMMs also suffer from routing congestion, thus limiting the

TABLE I: Comparison between existing GEMM architectures and our proposed uSystolic.

| Architecture | Accuracy | Power Efficiency | Scala-bility | Generali-zability |
|---|---|---|---|---|
| B-Systolic [30] | Precise | Low | High | High |
| B-Mesh [13] | Precise | Low | Low | High |
| FSU [54], [69], [75] | Low-High | High | Low | Low |
| HUB [38], [57], [58] | High | High | Low | Medium |
| uSystolic (ours) | High | High | High | High |

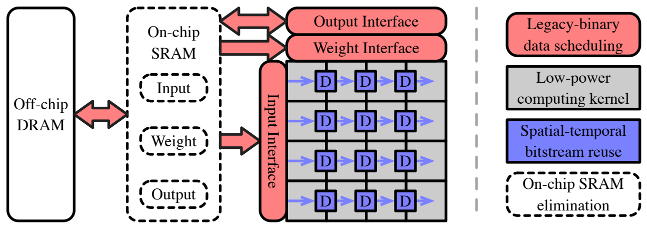


Fig. 1: uSystolic overview with memory hierarchy.

scalability. Furthermore, FSU architectures do not consistently offer high accuracy, e.g., temporal coding for signed data yields low accuracy [69]. On the other hand, there also exist hybrid unary-binary architectures (HUB in Table I) [15], [38], [45], [57], [58], which hybridize unary and binary computing to execute GEMM for better accuracy and higher scalability than FSU architectures. For example, [57] applies a 3-D mesh architecture to fit the inner-loops of matrix convolution. However, it is not well generalizable to matrix multiplication with inner-loops of varying shapes. As such, mapping arbitrary GEMMs on those architectures with no overhead is difficult.

***Proposed Low-Power Unary GEMM Architecture.*** To address above challenges, we propose uSystolic, a hybrid unary-binary systolic array architecture, which hooks unary computing kernels to binary interfaces. We highlight the features of uSystolic in Figure 1. First, uSystolic employs *legacy-binary data scheduling* at its memory interfaces. At the memory side, uSystolic exhibits identical data scheduling order as conventional binary systolic array and is broadly generalizable to varying GEMM configurations as shown in Table I. More specifically, uSystolic employs a high-accuracy unary multiplication [69], naturally coupling with weight stationary dataflow [30], [55]. This allows designers to integrate uSystolic to existing binary systems without sophisticated software reinnovation. Second, uSystolic internally equips *low-power computing kernels*, including low-cost unary multiplication and reduced-resolution binary accumulation. The signed multiplication in uSystolic is performed using the unsigned multiplier from [69] in the sign-magnitude format, i.e., the sign bit and the absolute value, halving the hardware cost and cycle count of the signed multiplier in [69]. The binary accumulation of bitstreams guarantees high accuracy [57], [58], especially for temporal-coded bitstreams of signed data, which exhibit poor accuracy if accumulated in unary domain [69]. The output resolution can also be reduced for extra hardware saving, compared to binary designs. Similar to [69], dynamic accuracy-energy trade-off via early termination [69], [72] is viable in uSystolic. Third, uSystolic is featured with *spatial-temporal bitstream reuse*, which transforms the input reuse in binary systolic arrays to unary domain. This bitstream reuse, embedded in the local interconnections, endues a much stronger scalability in uSystolic than prior FSU [69] or HUB [58] architectures relying on global broadcast and preserves consistently high accuracy. Lastly, *on-chip SRAM elimination* is feasible in uSystolic due to the relaxed requirement for high memory bandwidth. Though unary multiplication prolongs the latency, it slows down the data movement, i.e., data bytes are crawling out from memory to drive uSystolic, reducing the memory bandwidth, especially in edge scenarios.

TABLE II: GEMM parameters from the DNN perspective.

| **GEMM Parameter (Shape)** | | **Matrix Operation (Type)** | |
|---|---|---|---|
| | | **Convolution** | **Multiplication** |
| Input feature map (I) | Height | $IH$ | $IH$=1 |
| | Width | $IW$ | $IW$ |
| | Channel | $IC$ | $IC$=1 |
| Weight (W) | Height | $WH$ | $WH$=1 |
| | Width | $WW$ | $WW$ |
| | Stride | $S$ | $S$=1 |
| Output feature map (O) | Height | $OH$ | $OH$=1 |
| | Width | $OW$ | $OW$=1 |
| | Channel | $OC$ | $OC$ |

We list the contributions of this work as follows:

- This work identifies the technical challenges for low-power GEMM execution in existing literature as 1) on-chip SRAM and superquadratical power overhead in binary computing and 2) poor generalizability to diverse GEMM configurations in unary computing.
- This work presents uSystolic, a systolic array architecture that hybridizes low-power unary computing with binary interfaces, to overcome the deficiencies in existing unary GEMM designs by simultaneously offering high accuracy, efficiency, scalability and generalizability.
- We evaluate uSystolic at the system level in the context of GEMM-intensive DNNs using early termination as a key knob, and demonstrate that uSystolic feeds on crawling bytes, i.e., requires low bandwidth, to eliminate on-chip SRAM for even higher energy and power efficiency. Our customized systolic array simulator for evaluation, uSystolic-Sim, is publicly available [67].

The rest of this paper is organized as follows. Section II reviews the weight stationary systolic array and unary computing. Then, Section III describes the detailed architecture of uSystolic. Next, Section IV and V articulate the evaluation framework and result. Finally, Section VI concludes the paper.

## II. Background

This section reviews the systolic array with the weight stationary dataflow and the general concept and architectures of unary computing, all in the context of GEMM.

### A. Weight Stationary Systolic Array

GEMM operations are usually in the form of matrix multiplication, and can additionally be in the form of matrix convolution in the realm of DNNs [27]. We list the GEMM parameters, which unify the notation of both matrix convolution and multiplication, as in Table II. In this table, there exist three variables, including input feature map (IFM), weight, and output feature map (OFM), each associated with three parameters. Those variables are organized as data windows of shape (height, width), with the channel and stride referring to the number of windows and the stride to perform convolution on IFM and weight windows. The parameter values for matrix convolution and multiplication are directly taken from an ARM implementation of systolic arrays [55]. Using those parameters,

**Algorithm 1** Loop formulation for GEMM in Table II

```
1: For {oh,ow,oc,wh,ww,ic} in {OH,OW,OC,WH,WW,IC}
2:     O[oh,ow,oc] += W[oc,wh,ww,ic]
3:                 * I[wh+oh*S, ww+ow*S, ic]
4: return O
```

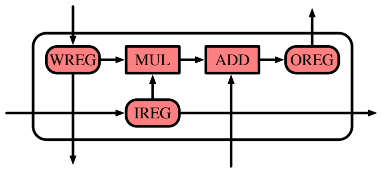


Fig. 2: Weight stationary binary systolic array PE.

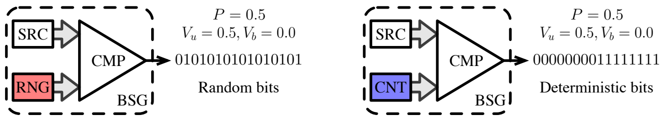
GEMM can be formulated as Algorithm 1, exposing abundant opportunities for data reuse to accelerate execution and boost efficiency [66]. There exist varieties of data reuse schemes, i.e., dataflows, including input stationary, weight stationary, output stationary dataflow, etc. [10], [55], [73]. In this work, we focus on the weight stationary dataflow, which is adopted by the commercialized Google TPU [30].

Weight stationary systolic arrays [30], [55] minimize the weight movement by reusing weights. The processing element (PE) using binary computing on bit-parallel data is in Figure 2. The IREG, WREG and OREG are used to store the IFM, weight and OFM (a partial sum), respectively. The multiplier (MUL) and adder (ADD) construct a single-cycle multiply-accumulate (MAC) unit. The weights are preloaded into the WREG and pipelined to bottom PEs. Then the IFMs are streamed into the IREG from left PEs to participate in MAC operations. The partial sum in OREG is accumulated to the OREG in top PEs for the final valid OFM. When all OFMs related to current weights are computed, new weights will be loaded. As high efficiency requires minimal stalls, binary systolic arrays put an exorbitant requirement on the memory bandwidth, which usually can't be offered by the off-chip DRAM. Therefore, binary systolic arrays need on-chip SRAM to allow local reuse for efficiency. In this work, uSystolic PE with unary computing takes multiple cycles, relieving the bandwidth requirement and unnecessitating the on-chip SRAM.

## B. Unary Computing

Unary computing's power supremacy originates from the simple computing units, which operate on serial bitstreams using either rate or temporal coding [16], [43], [69], with applications in error correction code [61], [68], image processing [1], [48], deep learning [54], [57], sorting [47], DNA sequencing [43], [44], superconducting [62], etc.

*1) Data Coding:* Unary data are serial bitstreams converted from parallel binary data. A $N$-bit binary data can be converted to a length-$2^N$ serial bitstream with an identical resolution. Unary bitstreams can employ varying codings [69], either *rate* [16] or *temporal* coding [43], as in Figure 3. Rate-coded and temporal-coded bitstreams, both representing their values by the probability of bit 1s, exhibit random and deterministic bit orders, respectively. The actual bitstream value not only



(a) Rate coding. (b) Temporal coding.

Fig. 3: Unary bitstream generation for rate and temporal coding by comparing SRC with a random and deterministic number sequence from SRC and CNT at each cycle. SRC: source binary data; RNG: random number generator; CNT: counter; CMP: comparator; BSG: bitstream generator.

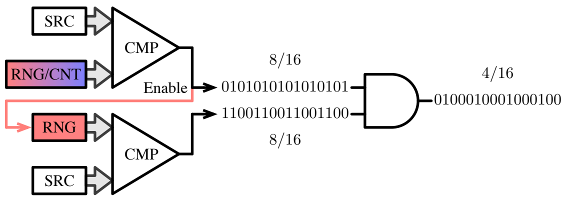
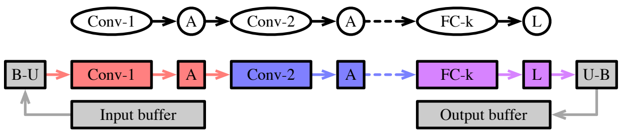


Fig. 4: Unipolar uMUL with conditional bitstream generation for zero *SCC*. Both inputs have the value of 0.5.

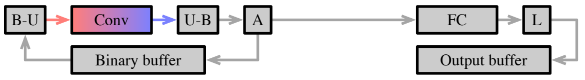
depends on the probability of bit 1s, i.e., the SRC value, but also relies on the polarity, either unipolar or bipolar. Given $P$ as the probability of bit 1s, the unipolar and bipolar bitstreams are unsigned and signed with values of $V_u = P$ and $V_b = 2 \cdot P - 1$, respectively. Note that the coding does not influence the data scheduling order in uSystolic.

*2) Unary Multiplication:* Unary computing can perform arithmetic operations, e.g., multiplication, with considerably cheap hardware [16] but exponentially increasing latency. A naive unary multiplication on unipolar bitstreams is an AND gate, exhibiting superquadratical area and power reduction compared with binary multipliers, which suffer from the routing congestion problem [68], [74]. For accurate unary multiplication [2], [69], [71], a zero-valued *stochastic cross correlation* (*SCC*) [2], formulated to measure the similarity between two bitstreams, is both necessary and sufficient. To force *SCC* close to zero, multiple solutions are proposed [21], [37], [42], [49], [69]. The recent uMUL calculates the product with high accuracy and low variance via conditional bitstream generation (*C-BSG*) [69], as illustrated in Figure 4. One bitstream is now the enable signal to update the RNG for another bitstream, whose source binary data are statically stored, i.e., stationary[1]. Consequently, the constraint of zero input *SCC* can be approximately converted to *C-BSG* between the enable bitstream (B) and the random sequence (R) from the RNG as in Equation 1. Furthermore, it is compatible to both rate and temporal coding. Due to those benefits, uSystolic will adopt *C-BSG* to ensure accuracy as in Section III-A. uSystolic multiplies two signed data in sign-magnitude format with one unipolar uMUL in Figure 4, which is only half the area of the bipolar uMUL in [69]. Furthermore, due to one less bit in the absolute value, only half the cycles are necessary for uSystolic,

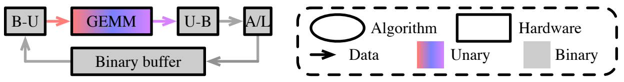
[1]This allows the dataflow to be either input or weight stationary, but not output stationary. We focus on weight stationary in this work.



(a) Existing fully streaming unary architecture.



(b) Existing hybrid unary-binary architecture.



(c) Proposed uSystolic architecture.

Fig. 5: Unary architectures for CNN execution. Binary and unary domains are gray and colored. B-U: binary data to unary bitstream conversion; U-B: unary bitstream to binary data conversion; GEMM: matrix convolution or multiplication; Conv: matrix convolution layer; FC: fully connected layer via matrix multiplication; A: activation; L: loss.

compared to [69], to perform signed unary multiplication.

$$SCC = 0 \cong C\text{-}BSG(B,R) \tag{1}$$

*3) Early Termination:* Though unary computing shows superquadratically higher power efficiency due to the simple logic, its energy efficiency is undetermined when the exponential latency overhead is considered. Prior research [39], [69] has unveiled that when the bitstreams are short enough, unary computing outperforms binary computing in energy efficiency. Thus, reducing the latency while not incurring significant accuracy loss is critical for the energy benefit, leading to the deployment of early termination, using technologies like output profiling [3], [68], hardware specialization [57] and metric-based characterization [69], [72]. Prior works [69], [72] have already proven the reliability of early terminating *C-BSG* with little accuracy loss for rate coding, while early terminating temporal-coded bitstreams incurs significant accuracy loss [69]. As such, in Section III-C, we will only present the architectural support for early termination in uSystolic for rate coding. Moreover, as early termination promotes the efficiency, we will use it as one key knob for evaluation in Section V.

*4) GEMM Architecture:* Based on the bitstream flow, unary GEMM designs can be categorized into fully streaming unary (FSU) and hybrid unary-binary (HUB) architectures.

*a) Fully Streaming Unary Architecture:* FSU architectures are fully parallel designs [40], [54], [69], [75]. An example of a k-layer DNN, e.g., convolutional neural networks (CNN), using a FSU architecture is given in Figure 5a. The input binary data are converted to unary bitstreams only once. The generated bitstreams then propagate through all hardware pipelines and are eventually converted back to binary domain at the output.

The recent uGEMM is a FSU GEMM architecture [69], [70], where high-performance unary multiplication and addition are organized together to perform unary GEMM operations, as shown in Figure 6. Input and weight bitstreams are first

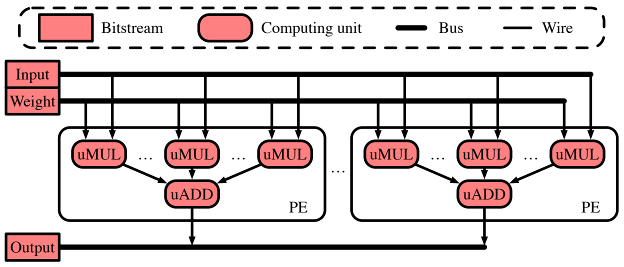


Fig. 6: SIMD FSU architecture in uGEMM [69].

broadcasted to the corespondent PE, and then multiplied and aggregated as a valid output bitstream using bipolar uMUL and uADD, respectively. All output bitstreams are computed simultaneously in uGEMM PEs, eliminating the sophisticated data scheduling in binary designs [9], [13], [30], [34], [51]. Such full parallelism classifies uGEMM as a SIMD architecture. Though uGEMM presents area and power advantages due to the simple unary logic, it has multiple drawbacks as in Table I. First, uGEMM has suboptimal output accuracy due to bitstream aggregation in the unary domain, especially for temporal coding [69], [70], despite accurate unary multipliers. Second, uGEMM exhibits low scalability due to broadcasts of input and weight bitstreams to all correspondent PEs via a global interconnection. Finally, uGEMM suffers low generalizability because it needs to be fitted for a fixed DNN model. To support multiple DNNs with different GEMM configurations, multiple uGEMM instances would be needed in hardware, diminishing the area and power advantages.

*b) Hybrid Unary-Binary Architecture:* To address the accuracy issue in uGEMM-like FSU architectures, HUB architectures with multiple rounds of intermediate interconversion between binary data and unary bitstreams [15], [38], [45], [57], [58] are proposed, with an example in Figure 5b. This HUB architecture executes matrix convolution with unary hardware, and performs the rest with binary logic. Its insight is to perform expensive operations with unary logic for efficiency and cheap operations with binary logic for accuracy.

However, existing HUB architectures are not as generalizable as the binary counterparts, as their dataflows [38], [57], [58] are not flexible enough for varying GEMM configurations. For example, authors in [38], [58] parallelize all computation for one single output element in matrix convolution. But when the matrix shape and type change, this hardware is not able to provide the best efficiency. Authors in [57] apply a 3-D mesh architecture to compute matrix convolution without co-optimizing it for matrix multiplication. Nevertheless, the 3-D architecture optimized for inner loops of matrix convolution does not naturally match the shape of inner loops of matrix multiplication as in Table II, lowering the efficiency.

In this work, we propose uSystolic to simultaneously offer high accuracy, efficiency, scalability and generalizability (Table I) by equipping the well-studied systolic arrays with unary computing, as in Figure 5c. Compared with uGEMM, whose unipolar uMUL is leveraged here, uSystolic exhibits significant advantages resulting from small yet deliberate, fundamental optimizations. First, the binary accumulation allows accurate MAC for temporal coding, which is impossible in uGEMM. Second, the unipolar uMUL on sign-magnitude formatted data halves the latency and power of bipolar uMUL in uGEMM, improving the energy efficiency by more than $2\times$. Third, the systolic nature with local interconnections, instead of global interconnections in uGEMM, ensures high scalability with no accuracy degradation and minimized area overhead, powered by the spatial-temporal bitstream reuse. Last, as a systolic array, uSystolic inherits the data scheduling from commercial designs, offering high generalizability for varying GEMMs, unlike uGEMM targeting a fixed configuration.

## III. uSystolic Architecture

The proposed uSystolic is similar to the Google TPU systolic array [30] in two aspects. First, they both have fixed-point (FXP) signed binary data at the input and output. Second, they both adopt the weight stationary dataflow with an identical scheduling order. These similarities minimize the effort to deploy uSystolic in low-power edge computing scenarios. However, there exist more architectural differences. First, uSystolic employs a low-power computing kernel via hybrid unary-binary (HUB) computing, including low-cost unary multiplication and reduced-resolution binary accumulation. Second, uSystolic introduces a novel subword-level spatial-temporal bitstream reuse to further reduce the hardware cost while offering a consistent accuracy guarantee. Third, uSystolic is able to trade off accuracy for energy with early termination, boosting the energy efficiency. Fourth, uSystolic requires the instruction set architecture (ISA) to be aware of the PE runtime (cycle count). Lastly, uSystolic allows on-chip SRAM elimination due to the reduced bandwidth requirement, further improving energy and power efficiency. From the system performance perspective, the above differentiators provide uSystolic with user-tunable accuracy, bandwidth, throughput, energy and power efficiency.

The uSystolic architecture is shown in Figure 7. The entire uSystolic is located in the middle, containing $R$ rows and $C$ columns of PEs. The surrounding FIFOs are in charge of synchronizing data as in [30]. In step ❶, the weights are preloaded into the array from the top and remain stationary in the array until all related output feature maps (OFMs) are calculated. In step ❷, the binary input feature maps (IFMs) are fed into the array from the left. In step ❸, the IFMs are converted to unary bitstreams and pipelined to right PEs. In step ❹, the unary OFM bitstreams are accumulated in binary domain and output from the top.

### A. Low-Power Computing Kernel

The left part of Figure 7 draws the PEs at the leftmost column. Both the weight and IFM, originally signed, are converted to sign-magnitude format, i.e., the sign bit and the absolute value. Such conversion is done once at the leftmost column and the upmost row with minimum overhead. The absolute values of the weight and IFM are then compared with the output of the RNG or CNT to generate unipolar bitstreams as in Figure 3. Weight bitstreams always apply rate coding using RNG, while IFM bitstreams can optionally be of temporal coding using CNT. The feedback signal from the IFM comparator C-I to the weight RNG acts as the enable signal in Figure 4, forming a unipolar uMUL for low-cost unary multiplication. Then the output of uMUL AND gate needs to be accumulated to the OREG, if the unary multiplication is not completed, i.e., M-end is not asserted. The accumulated value depends on the sign bits of both the weight and IFM, i.e., WSIGN and ISIGN: 1) if two bits are identified to be the same using the XOR gate, the uMUL output bit is directly accumulated; otherwise, its opposite is accumulated. When M-end is asserted, the partial sum in OREG from the PE below is accumulated into the current OREG, and finally streamed out as a valid OFM.

In above HUB computing flow (binary-unary-binary), a $N$-bit binary weight and a $N$-bit binary IFM first generates two $2^{N-1}$-bit unary bitstreams. Then two bitstreams produce one $2^{N-1}$-bit bitstream, which accumulates to be a $N$-bit binary OFM. This means applying this flow does not change the FXP data resolution at the input and output, in contrary to the fact that the conventional binary multiplication of two $N$-bit binary input produces one $2N$-bit binary output. This allows to optimize the PE with reduced-resolution binary accumulation: the OREG size can be $N$-bit smaller than that in binary designs, further reducing the hardware cost. The effect of resolution reduction is later evaluated in Section V-A. Note that long MAC cycles allow to better hide timing fluctuation of data synchronization in the FIFO, even without on-chip SRAM.

### B. Spatial-Temporal Bitstream Reuse

The right part of Figure 7 depicts the PEs located at the columns other than the leftmost. The low-power uMUL still exists, labelled in gray, and is simplified in two aspects: 1) the IFM bit to the uMUL is the delayed IFM bit in IDFF from the left PE; 2) the random number to generate the weight bit is the delayed version of that from the left PE in the RREG. Such an organization ensures that both the IFM bitstream and weight random number are generated only once and reused both spatially and temporally in a row. This spatial-temporal bitstream reuse allows to 1) to reduce the area and power overheads by eliminating two costly RNGs and one comparator, and 2) more importantly, to guarantee the consistency in high accuracy throughout the entire systolic array.

The PEs at the leftmost column employ uMUL to satisfy the zero-correlation requirement for high accuracy in Equation 1. Given $r$ and $c$ as the row and column indexes, leftmost PEs with index ($r$, 0) meet Equation 2. Such a relationship holds for a duplication of the bitstream $B_0$ and the random number sequence $R_0$, e.g., uGEMM spatially duplicates them via broadcasting [69]. In uSystolic, the one cycle lag for IDFF and RREG in a PE from those in the left PE duplicates the values both spatially and temporally, formulated in Equation 3. Combining Equation 2 and Equation 3, we have Equation 4, where $0 \leq r < R$ and $0 \leq c < C$, to indicate that all PEs at the same row follow the same *SCC* constraint for consistently high accuracy. Furthermore, applying the same RNG to all rows in

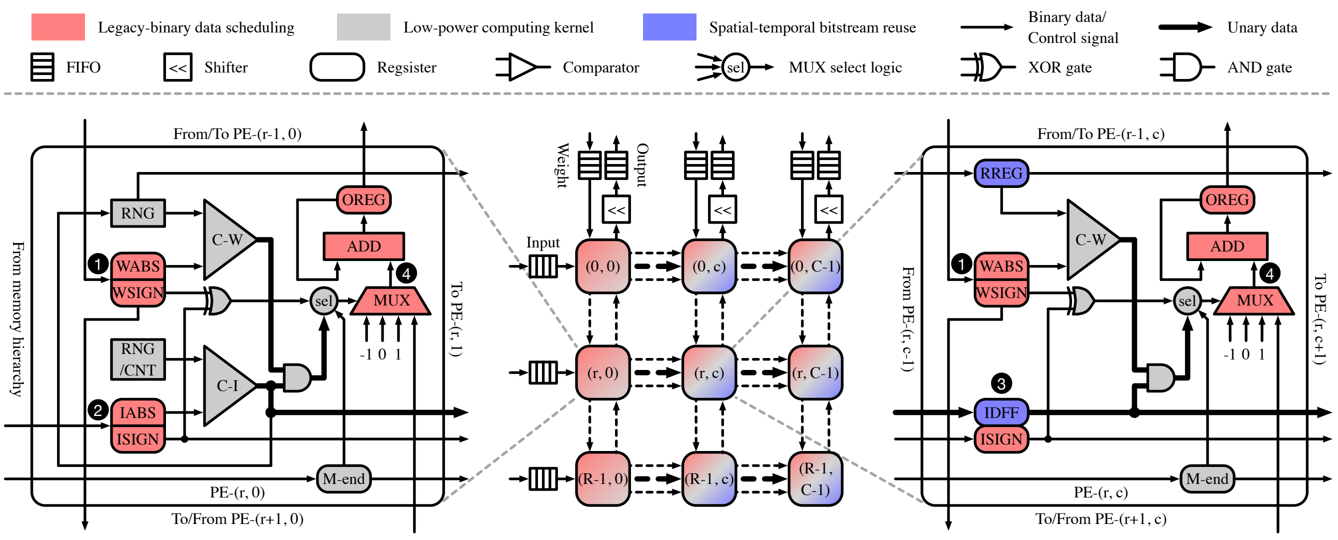


Fig. 7: uSystolic architecture with total $R$-by-$C$ PEs. In the middle is the array of PEs; on the left is the diagram of PEs at the leftmost column of the array; on the right is the diagram for the rest PEs in the array. WABS and WSIGN are registers to buffer the absolute value and sign of the weight; IABS and ISIGN are registers for the absolute value and sign of the input feature map; IDFF is to buffer the input feature map bit from the left PE. C-W and C-I are comparators to generate bitstreams for the weight and input feature map. RNG and CNT denote the random number generator and counter. RREG and OREG are buffers for the random number from the left PE and the output, respectively. M-end signals the end of unary multiplication.

uSystolic, we achieve an identical accuracy level throughout all PEs. And we configure the RNG in uSystolic to be the high-quality Sobol RNG [42] as in [69].

$$SCC = 0 \cong C\text{-}BSG_r(B_0, R_0) \tag{2}$$

$$C\text{-}BSG_r(B_c, R_c) \implies C\text{-}BSG_r(B_{c+1}, R_{c+1}) \tag{3}$$

$$SCC = 0 \cong C\text{-}BSG_r(B_c, R_c) \tag{4}$$

### C. Early Termination

The early termination in uSystolic is designed for rate coding to improve the energy and power efficiency, instead of temporal coding due to the potential accuracy loss as explained in Section II-B3. Given a $N$-bit binary weight and IFM, the binary product can be maximally $N$-bit if all unary bits are accumulated. Early termination leads to a fraction of those bits being accumulated. Assume $2^{n-1}$ bits are accumulated, the binary product will be $n$-bit, where $1 \leq n \leq N$. This fact implies that only $n$ out of $N$ most significant bits of the binary output are generated, and we define $n$ to be the *effective bitwidth*. The resultant MAC cycle count is $2^{n-1}+1$. This $n$-bit result needs to be scaled back to $N$ bits (left shifted by $N-n$) for correctness. Furthermore, the shifters can be allocated to the output of PEs in the top row, instead of per PE for hardware saving, as in the middle of Figure 7.

### D. ISA Support

As uSystolic maintains the dataflow of binary systolic arrays, it has an identical data scheduling order as in binary systolic arrays. However, uSystolic increases the PE MAC cycle count. Therefore, the interval between consecutive data scheduling is deterministically prolonged. More specifically, the weight preloading is identical to that in binary systolic arrays, while the original one-cycle OFM accumulation in Figure 2 now lags multiple cycles behind the initial streaming of IFM. As such, the uSystolic ISA is similar to that of TPU, but augmented with an indicator field for the MAC cycle count, i.e., how many cycles to terminate the computation.

### E. On-Chip SRAM Elimination

The memory hierarchy of uSystolic is used to store weights, IFMs and OFMs. However, the increased PE MAC cycle count in uSystolic leads to less frequent data feeding at the memory interfaces, i.e., data bytes are crawling out from the memory, reducing bandwidth for both on-chip SRAM and off-chip DRAM. Fortunately, the on-chip bandwidth can be as low as what off-chip DRAM can cheaply afford, and even some crawling bytes can drive uSystolic in edge computing scenarios. Therefore, the on-chip SRAM can be of small size, or even eliminated for energy and power benefits, without hurting data synchronization. In this work, we focus on the necessity of SRAM, as shown by the dashed blocks in Figure 1, instead of judiciously exploring how large SRAM needs to be.

## IV. Evaluation Framework

In this work, we focus on the computing kernel of the systolic array and memory hierarchy to understand where the limit of uSystolic lies. Therefore, other hardware logic in a comprehensive DNN accelerator, like accumulators, look-up tables, I/Os [30], are not evaluated. This section describes the evaluate framework in Figure 8 for the system in Figure 1 with both uSystolic and memory hierarchy.

### A. Objective

As uSystolic applies non-traditional HUB computing, two aspects are evaluated: 1) functional correctness, i.e., accuracy;

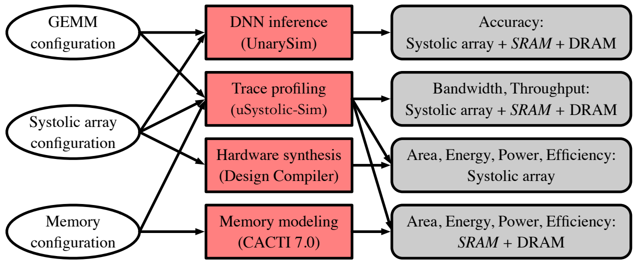

GEMM configuration
Systolic array configuration
Memory configuration
DNN inference (UnarySim)
Trace profiling (uSystolic-Sim)
Hardware synthesis (Design Compiler)
Memory modeling (CACTI 7.0)
Accuracy: Systolic array + SRAM + DRAM
Bandwidth, Throughput: Systolic array + SRAM + DRAM
Area, Energy, Power, Efficiency: Systolic array
Area, Energy, Power, Efficiency: SRAM + DRAM

Fig. 8: Evaluation framework of uSystolic. The objectives (right) are obtained by applying widgets (middle) on the configurations (left).

2) hardware performance , i.e., bandwidth, area, throughput, energy, power and correspondent energy and power efficiency in a system with the required memory hierarchy.

## B. Widget

To validate the accuracy of the system, we choose the popular GEMM-intensive DNN inference [27]. To model matrix convolution and multiplication behavior in uSystolic, we extend an open-source unary computing simulator, UnarySim [69], [70]. Besides the GEMM configuration in Table II, we also consider the data bitwidth and PE MAC cycle count.

The bandwidth and throughput are calculated by profiling memory traces modelled with a customized systolic array simulator, uSystolic-Sim [67], adapted from SCALE-Sim [55] from ARM. Our simulator supports varying computing schemes, different data bitwidths and memory-contention-aware data scheduling, which are unavailable in ARM SCALE-Sim.

The systolic array area and leakage power are modelled via hardware synthesis using Synopsys Design Compiler, while those for the memory hierarchy are modelled using CACTI 7 [6]. The dynamic power and energy of all are modelled with additional execution statistics from trace profiling. The efficiency is obtained by dividing the throughput by the energy and power. Note that the on-chip systolic array and SRAM always apply the same technology node.

## C. Configuration

*1) GEMM:* GEMMs are organized as CNN layers by configuring the shape and type in Table II. Such an organization is motivated by three reasons: 1) the accuracy of uSystolic can be directly reflected by GEMM-intensive CNN inference [22], [33], [59]; 2) among popular DNNs [7], [12], [24], [33], [59], [63], CNNs usually have the least ratio and simplest form of non-GEMM operations [46] to accurately reflecting the capability of uSystolic; 3) CNN layers exhibit varying GEMM configurations, covering representative use cases.

The involved CNNs and corresponding datasets are a small 4-layer CNN for small-sized MNIST dataset [36], the medium ResNet18 [22] for medium-sized CIFIAR10 dataset [32] and the large AlexNet [33] for large-sized ImageNet dataset [11]. Those CNNs have a parameter count of 1.2$M$, 11.7$M$ and 61.1$M$, respectively, representing small-, medium- and large-scale applications. The accuracy of all three CNNs are evaluated to proof the applicability, while only the layerwise performance of AlexNet is evaluated in detail.

To further demonstrate uSystolic's generalizability, we evaluate uSystolic's overall efficiency on the entire MLPerf benchmark [53], [55], a collection of diverse DNN models, including AlphaGoZero for Go match, AlexNet, GoogleNet and Resnet50 for image classification, neural collaborative filtering for recommendation system, sentimental_seqCNN and sentimental_seqLSTM for text sentiment analysis, and transformer for natural language processing, in total containing 1094 GEMM layers with varying configurations.

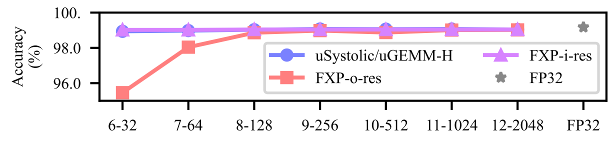
*2) Systolic Array:* The systolic array configuration refers to the shape, applied computing scheme, data bitwidth, PE MAC cycle count and dataflow. The choice of the configuration aims to highlight the impact of the computing scheme, instead of the rest. All systolic arrays are synthesized at 400$MHz$ under TSMC 32$nm$ technology to collect the hardware statistics.

According to the shape, represented by (height, width), systolic arrays are categorized into the edge configuration to mimic classical low-power use cases and the cloud configuration to extensively understand uSystolic's capability in high performance scenarios. The edge and cloud configurations take their shapes from MIT Eyeriss [10] (as small as (12, 14)) and Google TPU [30] (as large as (256, 256)), respectively, simplifying the design space exploration [4], [17], [29], [52].

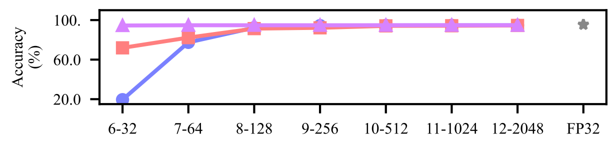
The evaluated computing schemes cover four versions: unary rate-coded, unary temporal-coded, binary bit-parallel and binary bit-serial, with each offering unique PE latency, area, energy and power. The two unary approaches are as in Section III. The binary bit-parallel implementation is the conventional binary version [30] with 1 cycle for a MAC operation. The binary bit-serial version is also evaluated here towards a holistic understanding between bit-serial binary computing and unary computing beyond the architecture level [69]. Its multiplication follows those in prior works [31], [56], where the MAC cycle count equals the data bitwidth plus 1 accumulation cycle, as only one MUL input in Figure 2 is serialized. We additionally evaluate a HUB systolic array with rate coding, uGEMM-H, based on [69] [2]. uGEMM-H implements the signed multiplication with the bipolar uMUL directly on the signed data, instead of the unipolar uMUL in uSystolic on sign-magnitude formatted data. Compared to uSystolic, uGEMM-H has no logic related to the absolute value; but uGEMM-H spends twice the area and cycles on signed unary multiplication to achieve identical binary accumulation resolution.

The data bitwidth is 8 from Eyeriss or 16 from TPU. For either, early termination for rate coding leads to different MAC cycle counts (i.e., the multiplication cycles plus 1 accumulation cycle). However, temporal coding allows no early termination for accuracy consideration as in Section II-B3.

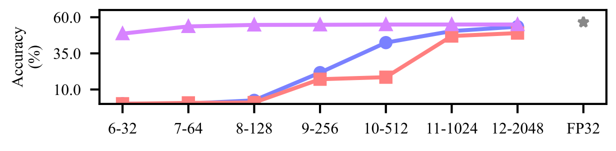
[2]The FSU architectures with rate coding in Table I, including the uGEMM [69], are not evaluated in this work for two reasons. First, AlexNet impractically requires 61.1$MB$ on-chip weight storage (D Flip Flops). Even more is needed for the entire MLPerf benchmark, far beyond the 24$MB$ SRAM in the Google's cloud TPU [30]. Second, temporal coding for signed data in FSU architectures incurs low DNN accuracy.



(a) MNIST dataset using a 4-layer CNN with 1.2$M$ parameters.



(b) CIFAR10 dataset using ResNet18 with 11.7$M$ parameters.



(c) ImageNet dataset using AlexNet with 61.1$M$ parameters.

Fig. 9: Top-1 accuracy comparison for CNN inference.

Finally, as uSystolic adopts a weight stationary dataflow, we fairly apply it to all designs, even Eyeriss for the edge, which is originally row stationary. Albeit switching the dataflow indeed differentiates the hardware performance from the original version, it does not deviate from our goal, i.e., evaluating the influence of varying computing schemes on performance.

*3) Memory Hierarchy:* SRAMs can be either present or absent for three variables in Table II. SRAMs, if existing, are double buffered to hide the access latency [55], [66], and their sizes are also originated from Eyeriss and TPU in the edge and cloud configurations. Both Eyeriss and TPU employ a shared global buffer for three variables, and we divide the entire SRAM space evenly into three equal pieces for three variables. Eyeriss has a global buffer of size 108$KB$ and a local scratchpad of size 0.5$KB$ for each PE, i.e., in total 192$KB$ on-chip SRAM, and we assign 64$KB$ to each variable in the edge configuration. TPU has a 24$MB$ global buffer and we assign 8$MB$ to each variable in the cloud configuration. Also, to reduce the SRAM bank conflict, we set the bank count of each variable SRAM in both configurations to 16. The SRAM technology node is aligned with that of the systolic array, i.e., 32$nm$. We assign the same 22nm 1$GB$ DDR3 DRAM chip as the off-chip memory in all on-chip configurations. The DRAM bank count is set to 8, and the page size is set to 8192 bits.

## V. Experimental Result

In this section, we summarize the results and insights for each objective using early termination a key evaluation knob.

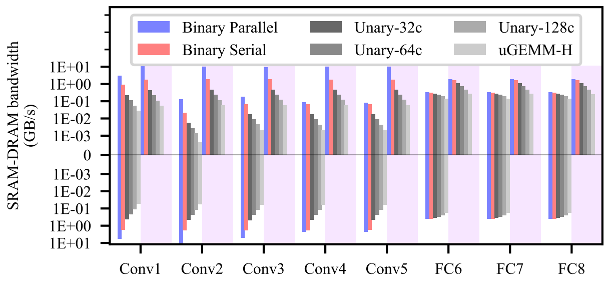
### A. Accuracy

The top-1 accuracy of CNN inference are shown in Figure 9. As uSystolic computes in a fundamentally different way from both floating-point (FP) and fixed-point (FXP) designs, we compare uSystolic against three binary designs, including one single-precision FP design, i.e., FP32, and two FXP designs, i.e., FXP-o-res and FXP-i-res. The FP32 design is the original model, while the rest are obtained from the FP32 design by simply quantizing all variables to FXP format. The accuracy evaluation focuses on revealing the gap between different computing schemes, thus, no accuracy-preserving retraining is performed. uSystolic applies varying *effective bitwidth* (EBT), defined as the actual number of bits, $n$, used in the HUB computing in Section III-C. We set $6 \leq n \leq 12$, as the accuracy either degrades too much or saturates outside this range. The EBT is marked on the horizontal axis, together with the correspondent multiplication cycle count in uSystolic, in the format of (*effective bitwidth*)-(*cycle count*). For example, 6-32 means unary multiplication in uSystolic has 6-bit EBT and needs 32 cycles, where $32 = 2^{6-1}$ due to unary multiplication is always performed on unipolar bitstreams as in Section III-A. FXP-o-res and FXP-i-res both apply conventional binary computing on FXP input data. FXP-o-res refers to that the output resolution is $n$ bits. If $n$ is even, two MAC inputs (weight and IFM) are both $\frac{n}{2}$ bits; otherwise, the weight/IFM are either $\frac{n-1}{2}/\frac{n+1}{2}$ or $\frac{n+1}{2}/\frac{n-1}{2}$ bits, whichever produces higher accuracy. FXP-i-res means the input resolution is $n$ bits, leading to $2n$-bit output resolution. As such, uSystolic with $n$ bits for both input and output has an intermediate accuracy between FXP-o-res and FXP-i-res, i.e., both the mean and standard deviation of the error for GEMMs rank as FXP-o-res $<$ uSystolic $<$ FXP-i-res as simulated. Note that the uSystolic accuracy for rate and temporal codings with an identical EBT are almost the same.

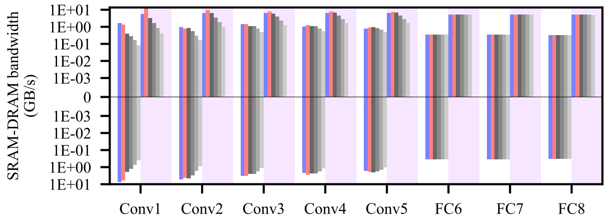
As in Figure 9, for all CNNs, FP32 has the highest accuracy, with FXP-i-res ranking the second across all EBT. For MNIST dataset using a 4-layer CNN, we barely see accuracy drop in uSystolic, which outperforms FXP-o-res. However, when the task complexity rises, i.e., the dataset becomes larger, the accuracy varies. When the EBT is 6 or 7, FXP-o-res has higher accuracy than uSystolic; but uSystolic consistantly excels for the rest EBT. For rate coding, smaller EBT can be obtained by early terminating larger EBT. Note that uGEMM-H has the same accuracy as uSystolic, as bipolar uMUL in uGEMM-H merely changes the hardware cost but not resolution of unary multiplication. One key observation is that *uSystolic presents smooth accuracy variation when the EBT changes across all tasks, translating to satisfactory accuracy-runtime scaling to boost the hardware efficiency via early termination.*

### B. Bandwidth

The GEMM-level (per layer in 8-bit AlexNet) SRAM and DRAM bandwidth of all candidate designs are presented in Figure 10. The bandwidth (and following throughput, energy and power) plots omit temporal coding, as they are similar to rate coding without early termination. For the edge, more multiplication cycles always decrease DRAM bandwidth and SRAM bandwidth if SRAM exists, due to the insignificant memory contention. Eliminating SRAM transmits the SRAM bandwidth pressure to DRAM, increasing the maximum DRAM bandwidth of binary parallel and serial designs from 3.03GB/s and 0.88GB/s to 10.49GB/s and 1.83GB/s. Though the uSystolic DRAM bandwidth also rises, it still maintains at a very low level, e.g., $[0.11, 0.47]$GB/s for compute-bounded Conv layers and $[0.46, 1.08]$GB/s for memory-bounded FC layers. The key insight is that *uSystolic requires ultra-low*



(a) Edge configuration.



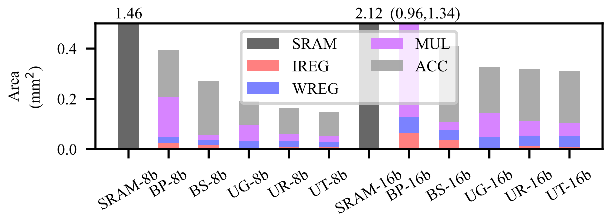
(b) Cloud configuration.

Fig. 10: **Log-scale** layerwise bandwidth comparison for 8-bit AlexNet. The upper and lower planes are for DRAM and SRAM if any. Conv and FC are convolution and fully connected layers. Binary Parallel and Binary Serial denote binary computing with 1 cycle for bit-parallel multiplication (also for MAC) and 8 cycles for bit-serial multiplication. Unary-32c, 64c and 128c refer to rate-coded unary computing with 32, 64 and 128 cycles for unipolar multiplication, where 32 and 64 cycles are early terminated from 128 cycles. uGEMM-H spends 256 cycles on bipolar multiplication with rate coding. White and purple background colors are for designs with/without on-chip SRAM.

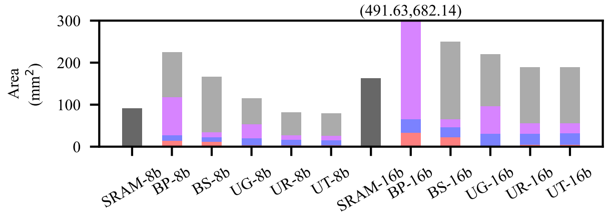
*DRAM bandwidth, even when SRAM is absent, allowing to eliminate the more-than-enough SRAM from uSystolic for edge computing*. uGEMM-H requires even lower bandwidth due to longer MAC cycles. In edge devices, such low bandwidth opens opportunities to drive the computation in uSystolic with only a few crawling DRAM bytes, which is impossible for binary designs. As such, in the following we focus on the hardware evaluation on binary designs with SRAM and unary designs without SRAM. In the cloud configuration, increasing the MAC cycle count does not monotonically decrease the bandwidth due to heavy memory contention.

## C. Area

The area breakdown of 8-bit and 16-bit systolic arrays, excluding the insignificant FIFOs and shifters in Figure 7, are presented in Figure 11. 16-bit designs have twice SRAM size of 8-bit designs to hold the same amount of data. When switching the computing scheme from BP to BS, UG, UR, UT for 8-bit designs, the systolic array area is observed to decrease by 30.9%, 50.9%, 59.0% and 62.5% for the edge and 26.2%, 48.9%, 63.8% and 64.7% for the cloud. IREG with spatial-temporal bitstream reuse, MUL with low-cost unary multiplication and spatial-temporal bitstream reuse, and ACC with reduced-resolution binary accumulation contributes 3.9%, 33.4% and 21.3% for rate-coded uSystolic with the edge configuration. Though BS designs have smaller MUL than



(a) Edge configuration.



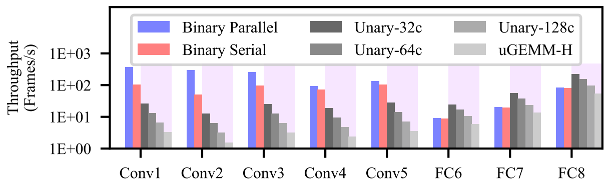
(b) Cloud configuration.

Fig. 11: **Linear-scale** area comparison. BP, BS, UG, UR and UT denote systolic implementations using binary parallel, binary serial, uGEMM-H rate, uSystolic rate and uSystolic temporal computing schemes, respectively. The -8b and -16b indicate the 8 and 16 bits for two multiplication inputs. The IREG, WREG and MUL directly refers to correspondent blocks in Figure 2 for binary computing, with ACC for ADD/OREG. In uSystolic from Figure 7, IREG is for the IABS/IDFF/ISIGN, WREG contains WABS/WSIGN, MUL includes RNG/CNT/RREG/C-W/C-I/AND, and ACC consists of the rest. The area of IREG, WREG, MUL and ACC are stacked as a whole to reflect the area of the entire systolic array. The outlier values are labelled on top of the correspondent bars, with the single value as the top of SRAM area and the value tuple as the top of MUL and ACC area.

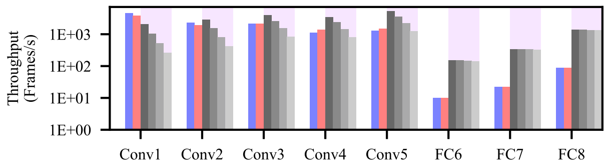
uSystolic, the overall area is higher due to larger ACC. The rate-coded uSystolic for the edge has 58.2% smaller MUL than uGEMM-H with bipolar uMUL, leading to 16.5% overall area reduction, though the remaining logic is larger due to the sign-magnitude data format. For 8-bit designs, rate-coded uSystolic without SRAM exhibits 91.3% and 90.7% area reduction for the edge and 74.3% and 68.4% area reduction for the cloud, from BP and BS designs with SRAM. More savings are observed for 16-bit designs. We conclude that *the proposed techniques in uSystolic all boost the area efficiency, among which the on-chip SRAM elimination contributes the most.*

## D. Throughput

The throughput, i.e., the reciprocal of the runtime, is compared in Figure 12. For Conv layers on the edge, the throughput degrades almost linearly when the MAC cycle count increases, due to low memory contention (2.7%, 1.3% and 0.7% average runtime overhead for 32-, 64- and 128-cycle uSystolic, and 0.3% for 256-cycle uGEMM-H). For Conv layers on the cloud, the throughput does not monotonically descend as MAC cycles increase, as the memory contention in binary parallel and serial designs introduces 161.8% and 105.2% average runtime overhead, while it's only 47.5%, 25.7% and 13.4% for 32-, 64- and 128-cycle uSystolic, and 6.9% for 256-cycle



(a) Edge configuration.



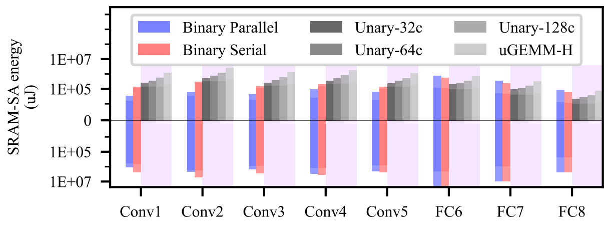
(b) Cloud configuration.

Fig. 12: **Log-scale** layerwise throughput comparison with identical notations as in Figure 10.

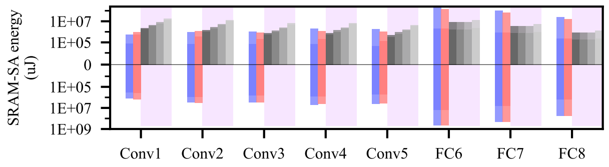
uGEMM-H, respectively. For both the edge and cloud, the FC throughput in uSystolic outperforms that in binary designs. The takeaway is that *in edge scenarios with little memory contention, early termination in uSystolic increases the throughput almost linearly with the reciprocal of MAC cycles.*

## *E. Energy*

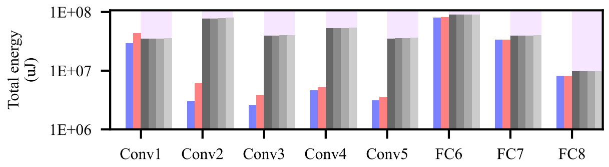
We plot the energy comparison in Figure 13. For binary parallel and serial designs for both the edge and cloud configurations, the SRAM leakage energy dominates the SRAM energy, which further dominates the on-chip energy. In comparison to the binary parallel and serial designs, uSystolic reduces the on-chip energy by $[50.0, 99.1]\%$ (mean 83.5%) and $[78.3, 99.1]\%$ (mean 90.5%) for the edge and $[-330.3, 98.9]\%$ (mean 47.6%) and $[-218.0, 98.8]\%$ (mean 55.5%) for the cloud. The negative gains for the cloud occurs for some Conv layers when the runtime increases, which scales up the dominant leakage energy. When considering the total energy including the off-chip DRAM dynamic access energy, the DRAM energy dominates. Compared with the binary parallel and serial designs, the total energy reduction in uSystolic ranges within $[-2474.7, -11.8]\%$ (mean $-754.0\%$) and $[-1147.1, 20.1]\%$ (mean $-487.1\%$) for the edge and $[-351.2, 92.3]\%$ (mean 18.1%) and $[-253.5, 91.5]\%$ (mean 25.1%) for the cloud, respectively. The negative gains mainly originate from matrix convolution, where the IFMs and OFMs in DRAM for matrix convolution are accessed much more frequently than those in matrix multiplication. uGEMM-H consistently consumes over $2\times$ energy than uSystolic, due to larger area and longer runtime. Overall, the energy consumption decreases in Figure 13 with shorter cycles, eventually causing the accuracy drop in Figure 9. When further considering the energy delay product (EDP), uSystolic does not provide improvements as high as those in the energy due to the exponential latency overhead. Compared to binary parallel and serial designs, the on-chip EDP improvements are $[-4611.4, 99.7]\%$ (mean $-487.8\%$) and $[-243.5, 99.7]\%$ (mean 3.9%) for the edge and $[-3776.4, 99.9]\%$ (mean $-145.2\%$) and $[-2250.7, 99.9]\%$ (mean $-56.2\%$) for the cloud. Such degradation also exists in the total EDP. Two takeaways are 1) *uSystolic reduces the on-chip energy most of the time, but cannot determine the DRAM-access-dominated total energy*; and 2) *early termination in uSystolic reduces the on-chip energy and achieves dynamic accuracy-energy scaling, as the leakage current dominates.*



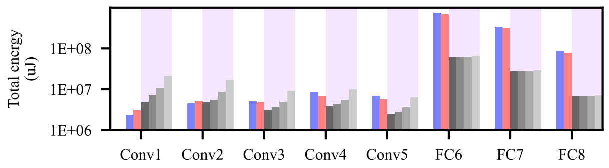
(a) Edge configuration (On-chip).



(b) Cloud configuration (On-chip).



(c) Edge configuration (Total: on-chip + off-chip).



(d) Cloud configuration (Total: on-chip + off-chip).

Fig. 13: **Log-scale** layerwise energy comparison. The upper and lower planes in (a) and (b) are systolic array (SA) and SRAM energy, whose sum is the on-chip energy. For each bar in (a) and (b), the brighter and dimmer parts (closer to and further from the x axis) represent the dynamic and leakage energy. (c) and (d) record total energy, including the on-chip energy and the DRAM dynamic access energy. The rest notations are identical to those in Figure 10.

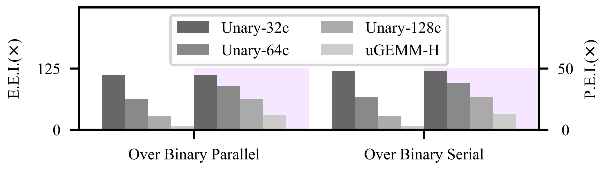
## *F. Power*

The layerwise power of each design can be calculated as the energy in Figure 13 by the throughput in Figure 12. For both the edge and cloud configurations, the SRAM leakage power of varying designs are identical, and dominate the SRAM power. The SRAM power for all evaluated GEMMs are larger than the systolic array power. Therefore, uSystolic exhibits tremendous on-chip power reduction, e.g., $[97.6, 99.5]\%$ (mean 98.4%) and $[97.5, 98.7]\%$ (mean 98.1%) for the edge and $[49.0, 83.4]\%$ (mean 66.4%) and $[48.7, 81.6]\%$ (mean 65.2%) for the cloud compared to binary parallel and serial designs. For both the edge and cloud configurations, early termination has varying influences on the dynamic power. For the edge configuration with low memory contention, less cycles raise the frequency to load data from memory into PE and increase the dynamic power. However, for the cloud, less cycles superlinearly increase the memory contention, as explained in Section V-D, and the resultant longer runtime inversely lowers systolic array dynamic power. Similar to energy, when further considering the DRAM dynamic access power, such colossal reduction is amortized, due to the dominance of DRAM dynamic access power. Again, compared with the binary parallel and serial designs, the total power reduction in uSystolic ranges in $[-220.2, 97.8]\%$ (mean $-37.0\%$) and $[-229.7, 94.9]\%$ (mean $-67.5\%$) for the edge and $[-48.0, 49.9]\%$ (mean $-6.1\%$) and $[-51.2, 52.2]\%$ (mean $-11.7\%$) for the cloud. The negative gains usually occur when the uSystolic MAC cycle count is small or the GEMM is matrix multiplication. Note that uGEMM-H always shows higher on-chip power than uSystolic due to larger area and longer runtime. Two lessons is that 1) *uSystolic indeed reduces the on-chip power tremendously, but a dedicated scheduling algorithm is mandatory to further reduce the off-chip DRAM access power*; and 2) *in edge scenarios with little memory contention, early termination in uSystolic will slight increase the on-chip power*.

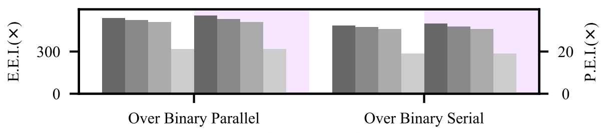
## G. Summary for Efficiency

Putting all together, accuracy-scalable uSystolic maintains the required bandwidth at a low level, where data bytes crawling out from DRAM at a low frequency can drive uSystolic. Based on this, we can safely eliminate the more-than-enough SRAM from uSystolic, which is not possible for binary systolic arrays. uSystolic reduces the on-chip area, energy and power tremendously due to the adopted techniques, especially eliminating the on-chip SRAM, which is usually costly and excels the systolic array in the hardware consumption. However, for the total energy and power, which are dominated by the off-chip DRAM, simply eliminating the SRAM does not bring about improvement, sometimes even degradation. Actually, eliminating SRAM leads uSystolic to the field of in-/near-memory computing [35], which appeals for specialized scheduling for high hardware performance. Above facts hold true for both edge computing and cloud computing use cases.

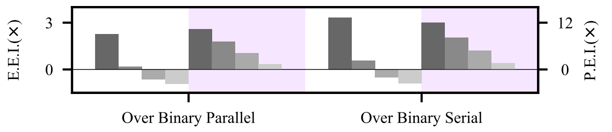
The on-chip energy and power efficiency (throughput/energy and throughput/power) gains of uSystolic over binary parallel and serial designs, running 8-bit AlexNet or MLPerf benchmark, are drawn in Figure 14. MLPerf yields lower efficiency than AlexNet, as diverse GEMMs reduce the average MAC utilization from 97.1% to 69.6% for the edge and from 81.6% to 37.2% for the cloud. The MLPerf energy efficiency on 128-cycle uSystolic and 256-cycle uGEMM-H for the edge is lower than that of binary designs, while all other efficiency is higher than that of binary designs. We observe that *early termination in uSystolic always increases the on-chip energy and power efficiency over binary designs, thanks to the increased throughput.* When considering the total energy and power with the DRAM access, such improvements almost vanish. Though we target the necessity of on-chip SRAM, there indeed exists a continuous design space where a small-sized on-chip SRAM can reduce the off-chip DRAM access cost.



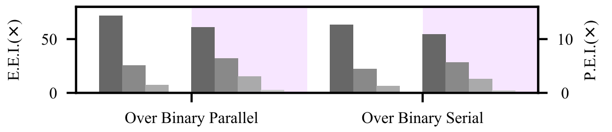
(a) Edge configuration for AlexNet.



(b) Cloud configuration for AlexNet.



(c) Edge configuration for MLPerf.



(d) Cloud configuration for MLPerf.

Fig. 14: **Linear-scale** on-chip energy and power efficiency improvement (denoted as E.E.I. (white background) and P.E.I. (purple background)) of uSystolic over binary parallel and serial systolic arrays. Each bar is the mean improvement for all GEMM layers in 8-bit AlexNet or MLPerf benchmark. The involved notations are identical to those in Figure 10.

## H. System-Level Discussion

The aforementioned architectural features in uSystolic can also be translated into dynamic trade-offs at the system level. When considering a single uSystolic instance, higher-quality (higher-accuracy) services can be offered at the cost of longer runtime with lower system bandwidth, i.e., not interrupting other components enormously. If the power supply, e.g., battery in edge computing, is running out, early termination improves energy and power efficiency to prolong the system lifespan. When considering multiple tiled uSystolic instances with interconnections, uSystolic's low bandwidth empowers better scalability. Moreover, its simple runtime control can hide packet routing variation in the interconnection, permitting flexible synchronization among instances.

## VI. Conclusion

This paper explores the bottleneck of low-power GEMM architectures for edge computing and presents uSystolic, a systolic array architecture based on unary computing. uSystolic employs a hybrid unary-binary architecture to utilize legacy-binary data scheduling and achieve low power consumption simultaneously via the low-power computing kernel, spatial-temporal bitstream reuse and on-chip SRAM elimination due to low bandwidth requirement with crawling data bytes. Experiments demonstrate that with 91.3% less on-chip area, uSystolic consumes 83.5% and 98.4% less on-chip energy and power, and exhibits up to 112.2× and 44.8× higher correspondent efficiency than binary designs for AlexNet.

## Acknowledgments

We thank our reviewers for their valuable feedback. This work is supported by the Wisconsin Alumni Research Foundation and NSF under award No. CNS-2045985.

## References

[1] A. Alaghi, Cheng Li, and J. P. Hayes, "Stochastic Circuits for Real-Time Image-Processing Applications," in *Design Automation Conference*, 2013.

[2] A. Alaghi and J. P. Hayes, "Exploiting Correlation in Stochastic Circuit Design," in *International Conference on Computer Design*, 2013.

[3] A. Alaghi and J. P. Hayes, "Fast and Accurate Computation Using Stochastic Circuits," in *Design, Automation & Test in Europe Conference & Exhibition*, 2014.

[4] C. Angermueller, D. Belanger, A. Gane, Z. Mariet, D. Dohan, K. Murphy, L. Colwell, and D. Sculley, "Population-Based Black-Box Optimization for Biological Sequence Design," in *International Conference on Machine Learning*, 2020.

[5] P. Aspinall, P. Mavros, R. Coyne, and J. Roe, "The Urban Brain: Analysing Outdoor Physical Activity with Mobile EEG," *British Journal of Sports Medicine*, vol. 49, no. 4, 2015.

[6] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories," *ACM Transactions on Architecture and Code Optimization*, vol. 14, no. 2, Jun. 2017.

[7] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational Inductive Biases, Deep Learning, and Graph Networks," *arXiv*, 2018.

[8] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning," in *USENIX Symposium on Operating Systems Design and Implementation*, 2018.

[9] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2014.

[10] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Conference on Computer Vision and Pattern Recognition*, 2009.

[12] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding," in *Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

[13] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting Vision Processing Closer to The Sensor," in *International Symposium on Computer Architecture*, 2015.

[14] V. V. Elango, N. N. Rubin, M. M. Ravishankar, H. H. Sandanagobalane, and V. V. Grover, "Diesel: DSL for Linear Algebra and Neural Net Computations on GPUs," in *International Workshop on Machine Learning and Programming Languages*, 2018.

[15] S. A. Faraji, G. Singh, and K. Bazargan, "HBUNN - Hybrid Binary-Unary Neural Network: Realizing A Complete CNN on An FPGA," in *International Conference on Computer Design*, 2019.

[16] B. R. Gaines, *Stochastic Computing Systems*. Springer, 1969, pp. 37–172.

[17] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google Vizier: A Service for Black-Box Optimization," in *International Conference on Knowledge Discovery & Data Mining*, 2017.

[18] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang, and W. J. Dally, "ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA," in *International Symposium on Field-Programmable Gate Arrays*, 2017.

[19] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *International Symposium on Computer Architecture*, 2016.

[20] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array Programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[21] M. Hassan Najafi, D. Jenson, D. J. Lilja, and M. D. Riedel, "Performing Stochastic Computation Deterministically," in *IEEE Transactions on Very Large Scale Integration Systems*, vol. 27, no. 12, 2019, pp. 2925–2938.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Conference on Computer Vision and Pattern Recognition*, 2016.

[23] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, R. Pang, Q. Liang, D. Bhatia, Y. Shangguan, B. Li, G. Pundak, K. C. Sim, T. Bagby, S. yiin Chang, K. Rao, and A. Gruenstein, "Streaming End-to-End Speech Recognition for Mobile Devices," in *International Conference on Acoustics, Speech and Signal Processing*, 2019.

[24] G. Hinton, S. Sabour, and N. Frosst, "Matrix Capsules with EM Routing," in *International Conference on Learning Representations*, 2018.

[25] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv*, 2017.

[26] H. Jebelli, M. M. Khalili, and S. Lee, "Mobile EEG-based Workers' Stress Recognition by Applying Deep Neural Network," in *Advances in Informatics and Computing in Civil and Construction Engineering*, 2019.

[27] Y. Jia, "Learning Semantic Image Representations at A Large Scale," Ph.D. dissertation, EECS Department, University of California, Berkeley, 2014. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-93.html

[28] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," in *International Conference on Multimedia*, 2014.

[29] A. Jones, A. Yazdanbakhsh, B. Akin, C. Angermueller, J. P. Laudon, K. Swersky, M. Hashemi, R. Narayanaswami, S. Chatterjee, and Y. Zhou, "Apollo: Transferable architecture exploration," in *Conference on Neural Information Processing Systems*, 2020.

[30] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. L. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. Richard Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-Datacenter Performance Analysis of A Tensor Processing Unit," in *International Symposium on Computer Architecture*, 2017.

[31] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-Serial Deep Neural Network Computing," in *International Symposium on Microarchitecture (MICRO)*, 2016.

[32] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," 2009.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Conference on Neural Information Processing Systems*, 2012.

[34] H.-T. Kung, "Why Systolic Architectures?" *Computer*, vol. 15, no. 1, pp. 37–46, Jan 1982.

[35] Y. C. Kwon, S. H. Lee, J. Lee, S. H. Kwon, J. M. Ryu, J. P. Son, O. Seongil, H. S. Yu, H. Lee, S. Y. Kim, Y. Cho, J. G. Kim, J. Choi, H. S. Shin, J. Kim, B. Phuah, H. Kim, M. J. Song, A. Choi, D. Kim, S. Kim, E. B. Kim, D. Wang, S. Kang, Y. Ro, S. Seo, J. Song, J. Youn, K. Sohn, and N. S. Kim, "25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with A 1.2TFLOPS Programmable Computing

Unit Using Bank-Level Parallelism, for Machine Learning Applications," in *International Solid-State Circuits Conference*, vol. 64, 2021, pp. 350–352.

[36] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[37] V. T. Lee, A. Alaghi, and L. Ceze, "Correlation Manipulating Circuits For Stochastic Computing," in *Design, Automation & Test in Europe Conference & Exhibition*, 2018.

[38] V. T. Lee, A. Alaghi, J. P. Hayes, V. Sathe, and L. Ceze, "Energy-Efficient Hybrid Stochastic-Binary Neural Networks for Near-Sensor Computing," in *Design, Automation & Test in Europe Conference & Exhibition*, 2017.

[39] V. T. Lee, A. Alaghi, R. Pamula, V. S. Sathe, L. Ceze, and M. Oskin, "Architecture Considerations for Stochastic Computing Accelerators," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, 2018, pp. 2277–2289.

[40] Z. Li, J. Li, A. Ren, R. Cai, C. Ding, X. Qian, J. Draper, B. Yuan, J. Tang, Q. Qiu, and Y. Wang, "HEIF: Highly Efficient Stochastic Computing-based Inference Framework for Deep Neural Networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 8, pp. 1543–1556, 2019.

[41] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen, "Cambricon: An Instruction Set Architecture for Neural Networks," in *International Symposium on Computer Architecture*, 2016.

[42] S. Liu and J. Han, "Toward Energy-Efficient Stochastic Circuits Using Parallel Sobol Sequences," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 26, no. 7, pp. 1326–1339, 2018.

[43] A. Madhavan, T. Sherwood, and D. Strukov, "Race Logic: A Hardware Acceleration for Dynamic Programming Algorithms," in *International Symposium on Computer Architecture*, 2014.

[44] A. Madhavan, T. Sherwood, and D. Strukov, "A 4-$mm^2$ 180-*nm*-CMOS 15-Giga-Cell-Updates-per-Second DNA Sequence Alignment Engine based on Asynchronous Race Conditions," in *Custom Integrated Circuits Conference*, 2017.

[45] G. Maor, X. Zeng, Z. Wang, and Y. Hu, "An FPGA Implementation of Stochastic Computing-based LSTM," in *International Conference on Computer Design*, 2019.

[46] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *International Conference on Machine Learning*, 2010.

[47] M. H. Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan, "Low-Cost Sorting Network Circuits Using Unary Processing," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 26, no. 8, pp. 1471–1480, 2018.

[48] M. H. Najafi and M. E. Salehi, "A Fast Fault-Tolerant Architecture for Sauvola Local Image Thresholding Algorithm Using Stochastic Computing," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 24, no. 2, pp. 808–812, 2016.

[49] F. Neugebauer, I. Polian, and J. P. Hayes, "Building A Better Random Number Generator For Stochastic Computing," in *Euromicro Conference on Digital System Design*, 2017.

[50] J. Park, Y. Boo, I. Choi, S. Shin, and W. Sung, "Fully Neural Network based Speech Recognition on Mobile and Embedded Devices," in *Conference on Neural Information Processing Systems*, 2018.

[51] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernandez-Lobato, G. Y. Wei, and D. Brooks, "Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators," in *International Symposium on Computer Architecture*, 2016.

[52] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized Evolution for Image Classifier Architecture Search," in *AAAI Conference on Artificial Intelligence*, 2019.

[53] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "MLPerf Inference Benchmark," in *International Symposium on Computer Architecture*, 2020.

[54] A. Ren, Z. Li, C. Ding, Q. Qiu, Y. Wang, J. Li, X. Qian, and B. Yuan, "SC-DCNN: Highly-Scalable Deep Convolutional Neural Network Using Stochastic Computing," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017.

[55] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "SCALE-Sim: Systolic CNN Accelerator Simulator," *arXiv*, 2018.

[56] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network," in *International Symposium on Computer Architecture*, 2018.

[57] H. Sim and J. Lee, "A New Stochastic Computing Multiplier with Application to Deep Convolutional Neural Networks," in *Design Automation Conference*, 2017.

[58] H. Sim, D. Nguyen, J. Lee, and K. Choi, "Scalable Stochastic-Computing Accelerator for Convolutional Neural Networks," in *Asia and South Pacific Design Automation Conference*, 2017.

[59] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations*, 2015.

[60] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J. S. Seo, and Y. Cao, "Throughput-Optimized OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks," in *International Symposium on Field-Programmable Gate Arrays*, 2016.

[61] S. S. Tehrani, W. J. Gross, and S. Mannor, "Stochastic Decoding of LDPC Codes," *IEEE Communications Letters*, vol. 10, no. 10, pp. 716–718, 2006.

[62] G. Tzimpragos, D. Vasudevan, N. Tsiskaridze, G. Michelogiannakis, A. Madhavan, J. Volk, J. Shalf, and T. Sherwood, "A Computational Temporal Logic for Superconducting Accelerators," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020.

[63] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention Is All You Need," in *Conference on Neural Information Processing Systems*, 2017.

[64] S. I. Venieris and C. S. Bouganis, "Latency-Driven Design for FPGA-based Convolutional Neural Networks," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2017.

[65] R. Wei, L. Schwartz, and V. Adve, "DLVM: A Modern Compiler Infrastructure for Deep Learning Systems," in *International Conference on Learning Representations*, 2018.

[66] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated Systolic Array Architecture Synthesis for High-Throughput CNN Inference on FPGAs," in *Design Automation Conference*, 2017.

[67] D. Wu, "uSystolic-Sim." [Online]. Available: https://github.com/diwu1990/uSystolic-Sim

[68] D. Wu, Y. Chen, Q. Zhang, Y.-L. Ueng, and X. Zeng, "Strategies for Reducing Decoding Cycles in Stochastic LDPC Decoders," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 9, pp. 873–877, 2016.

[69] D. Wu, J. Li, R. Yin, H. Hsiao, Y. Kim, and J. S. Miguel, "uGEMM: Unary Computing Architecture for GEMM Applications," in *International Symposium on Computer Architecture*, 2020.

[70] D. Wu, J. Li, R. Yin, H. Hsiao, Y. Kim, and J. San Miguel, "uGEMM: Unary Computing for GEMM Applications," *IEEE Micro*, vol. 41, no. 3, pp. 50–56, 2021.

[71] D. Wu and J. San Miguel, "In-Stream Stochastic Division and Square Root via Correlation," in *Design Automation Conference*, 2019.

[72] D. Wu, R. Yin, and J. San Miguel, "Normalized Stability: A Cross-Level Design Metric for Early Termination in Stochastic Computing," in *Asia and South Pacific Design Automation Conference*, 2021.

[73] X. Yang, M. Gao, J. Pu, A. Nayak, Q. Liu, S. E. Bell, J. O. Setter, K. Cao, H. Ha, C. Kozyrakis, and M. Horowitz, "DNN Dataflow Choice Is Overrated," *arXiv*, 2018.

[74] C. Yu and Z. Zhang, "Painting on Placement: Forecasting Routing Congestion Using Conditional Generative Adversarial Nets," in *Design Automation Conference*, 2019.

[75] J. Yu, K. Kim, J. Lee, and K. Choi, "Accurate and Efficient Stochastic Computing Hardware for Convolutional Neural Networks," in *International Conference on Computer Design*, 2017.

[76] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," in *Conference on Computer Vision and Pattern Recognition*, 2018.

[77] Y. Zhang, B. Du, L. Zhang, and J. Wu, "Parallel DNN Inference Framework Leveraging A Compact RISC-V ISA-based Multi-Core System," in *International Conference on Knowledge Discovery & Data Mining*, 2020.