Q-PIM: A Genetic Algorithm based Flexible DNN Quantization Method and Application to Processing-In-Memory Platform

Yun Long, Edward Lee, Daehyun Kim and Saibal Mukhopadhyay
yunlong, elee359, daehyun.kim@gatech.edu, saibal.mukhopadhyay@ece.gatech.edu

***Abstract*—This paper presents a genetic algorithm (GA) based training free layer-wise quantization method, named as GAQ, to reduce model complexity of arbitrary DNN architectures. The proposed algorithm formulates an optimization problem to determine the quantization level for each DNN layer under the constrain of maximum accuracy degradation and uses genetic algorithm to solve the problem at the inference stage of any pre-trained DNN models. The experimental results on various DNNs for image classification demonstrate 5x to 17x weight compression rate with insignificant ($< 2\%$) accuracy loss, comparable with existing quantization algorithms which typically require multi-pass retraining and handcrafted tuning. To evaluate the computational benefits of GAQ, we present a SRAM based flexible precision all-digital processing-in-memory (PIM) architecture, named as Q-PIM, that leverages GAQ to optimally control precision for each DNN layer to enhance efficiency. The simulation in 28nm CMOS shows potential for significant energy and latency advantage over fixed-precision PIM architectures.**

I. Introduction

The parameter and activation quantization has been explored as a promising technique to reduce the complexity of deep neural network (DNN) models and can be particularly beneficial for resource-constrained platform such as mobile devices, IoT edge devices, and other real-time systems. Recent DNN quantization works have demonstrate very promising compression rate with insignificant accuracy loss (i.e. less than 2% drop on ImageNet) [1]–[7]. But the existing techniques suffer from lack of quantization granularity, need for retraining, and low flexibility as discussed below.

Coarse quantization granularity. It has been observed that each layer of a given model has varying sensitivity to quantization [5]. However, most of existing works either employ uniform quantization or semi-uniform quantization, for example, quantizing most layers to a predefined precision and employing floating-point for rest of the layers [1].

Training requirement. Most of the existing quantization approaches require single or multiple passes of training. The training of large DNN models is not only computationally expensive, but also requires access to large-scale training dataset. Obtaining such dataset for domain specific tasks such as autonomous driving can be expensive as well as vulnerable to data privacy/security challenges.

Inflexible algorithms with handcrafted tuning. Existing DNN quantization algorithms rely on empirical method to determine precision reduction. For example, Apprentice [1] reserves the precision of the first/final layers in floating point and lowers the precision of the rest layers to 2-bit. PACT [6] keeps the shortcut convolution layer (Conv layer) of ResNet in floating point and quantizes other layers to 2-bit. Unfortunately, empirical (handcrafted) tuning of one DNN model may not guarantee performance for a different DNN.

This work was supported in part by the National Science Foundation (NSF-E2CDA #1740197) and in part by the Semiconductor Research Corporation.

This paper presents ***a genetic algorithm (GA) based and re-training free layer-wise quantization method, named as GAQ, to reduce model complexity of arbitrary DNN architectures.*** DNN quantization is treated as an optimization problem where the objective is to automatically decide a quantization level for each layer, under the constrain of a accuracy loss. The developed optimization problem is solved using genetic algorithm, a well-known technique in artificial intelligence. GAQ is re-training free and only works on inference stage with a small evaluation dataset. Moreover, the GAQ ensures that layer-wise quantization is performed in a fully automated manner (without handcrafted tuning) for any DNN architecture, making it flexible and scalable.

While GAQ is a hardware-agnostic algorithm, we demonstrate the performance and energy-efficiency impact of GAQ on a DNN accelerator considering Processing-In-Memory (PIM) platform [8]–[10]. Specially, we present ***a SRAM based flexible precision PIM inference accelerator, named as Q-PIM, that uses GAQ to optimally control precision for different DNN layers***. The Q-PIM adapts the in-memory bit accumulation based all-digital PIM designs [10] with redesigned memory peripherals and system architecture to enable computation with tunable (4/8/16/32-bit) precision.

We evaluate GAQ across various DNN models with different dataset, resulting to $5\times$ - $17\times$ weight compression rate with $< 2\%$ accuracy loss. We implement Q-PIM with 28nm CMOS, demonstraing the state-of-the-art computing efficiency. Moreover, algorithm-hardware co-simulation shows that compared to 16-bit fixed point precision running on Q-PIM platform, the Q-PIM with GAQ results in $2.8\times$ to $13\times$ lower latency and $3.3\times$ to $17\times$ lower energy for benchmarks DNNs including LeNet, AlexNet, VGG, and ResNet-18/34/50.

II. Background

A. DNN quantization algorithms

Early stage of the research on DNN quantization algorithms suffers from large accuracy drop when testing on deep mod-

978-1-7281-1085-1/20/$31.00 ©2020 IEEE

Authorized licensed use limited to: Georgia Institute of Technology. Downloaded on January 19,2023 at 15:20:33 UTC from IEEE Xplore. Restrictions apply.

els and large dataset. Recent works minimize the accuracy degradation. For example, Deep-compression [7] reduces the bit-precision of Conv layer and fully-connected layer to 8-bit and 5-bit, respectively. Combining with pruning, Deep-compression achieves negligible accuracy loss for AlexNet on ImageNet. Apprentice [1] leverages the knowledge distillation techniques and reduces the bit-precision of intermediate layer to 2-bit weight and 8-bit activation, resulting to 2.1% accuracy loss for ResNet on ImageNet. LQ-NETs [3] employs an adaptive quantization strategy, shows 0.3% accuracy loss for ResNet with 4-bit weight and floating point activation. Several other works, such as UNIQ [4], and PACT [6], also demonstrates good compression rate with insignificant accuracy drop. However, most of them requires re-training, large scale training dataset, additional trainable parameters, and handcrafted tuning. Moreover, empirical precision tuning for each layer to achieve layer-wise quantization is prohibitive for very deep models and can't guarantee the optimal tradeoff between accuracy and compression. An automated, flexible, and efficient quantization algorithm is still missing.

B. Basics of Genetic Algorithm

Genetic algorithm is a widely used heuristic search and optimization algorithm inspired by the process of natural selection and evolution. GA starts from an initialization process where a set of initial populations (i.e. candidate solutions for the targeting problem) are generated. Next an iterative process is used to evolve the initial population to an optimal solution. Given a target problem, a fitness function is used to evaluate the effectiveness of the candidates. Candidates with low fitness value are eliminated while candidates with high fitness value are kept to generate a new generation via crossover and mutation (i.e. re-generation). This process is repeated until a satisfactory solution is reached.

C. PIM Architecture

The PIM architecture using SRAM, DRAM, resistive RAM (ReRAM), and ferroelectric FET (FeFET) have been explored as promising solutions for energy-efficient acceleration of DNN [8]–[12]. Majority of the prior PIM accelerators explore analog computations using bit-line current summation, requiring data conversion between internally analog and externally digital domain. Since DAC and ADC introduce power/latency overheads and increases design complexity, digital PIM configurations have been proposed, including *in-memory logic* [9], [11], [12] and *in-memory bit accumulation* [10]. The all-digital design eliminates the need for ADC/DAC and demonstrates good energy-efficiency, specifically, for higher precision operations. In particular, we consider the all-digital PIM based on in-memory bit accumulation as our baseline design [10]. In this configuration, the array is accessed in row-by-row (sequential) manner; the AND operation (i.e. 1-bit multiplication) is performed between the word-line input data and the bit value stored in the cell. Digital accumulation is performed after the sense-amplifier for each clock cycle.

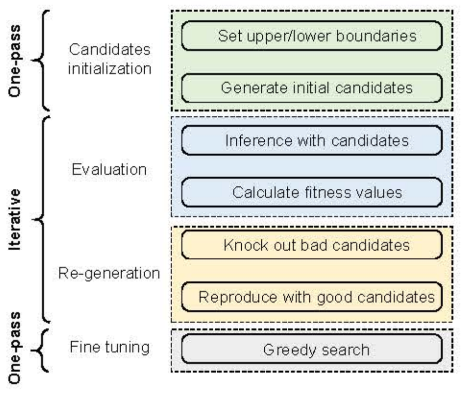

One-pass
Candidates initialization
Set upper/lower boundaries
Generate initial candidates
Iterative
Evaluation
Inference with candidates
Calculate fitness values
Re-generation
Knock out bad candidates
Reproduce with good candidates
One-pass
Fine tuning
Greedy search

Fig. 1. The overall GAQ flow.

Essentially, bit accumulation performs multiplication using multiple consecutive accumulation operation.

III. GAQ based DNN layer-wise quantization

Figure 1 shows the proposed GAQ flow. For simplicity, we use LeNet, a 5 layer CNN containing 3 conv layers and 2 fully-connected layer as an example. Parameter quantization (conversion from floating point to N-bit fixed point) in GAQ follows the same rule in existing works, namely, finding the parameter range and perform truncation evenly [1], [5]. GAQ is implemented in 4 steps.

Step ➊: Candidates initialization. Each initial candidate in GAQ is a vector containing N integers where N equals to the number of layers in the target DNN model and each number in that vector represents the bitwidth of a corresponding layer. For example, with LeNet, a possible candidate could be [8 8 8 5 5], indicating that 3 Conv layers are quantized to 8-bit and 2 fully-connected layers are quantized to 5-bit. To ensure the convergence speed, we first constrain the search space, namely, define the upper bound (UB) and lower bound (LB) for the GA exploration. The upper bound is defined in a heuristic way, for example, upper bound is defined as [8 8 8 8 8] for LeNet since 8-bit is good enough to guarantee the accuracy of LeNet on MNIST dataset. The lower bound is determined by the sensitivity of each layer. As shown in Figure 2 (a), the lower bound precision for a layer is defined as the bit-width that, once quantized, violates the accuracy constrain. All other layers are reserved in floating point precision when evaluating a given layer's sensitivity. For LeNet, the lower bound is set to be [1 2 1 2 1] under a $< 2\%$ accuracy drop constrain. With the upper/lower bound, an initial candidate can be generated with equation 1.

$$C = [rand(high = UB_i, low = LB_i) \;\; \forall i \in [0...N]] \tag{1}$$

$rand$ is the function to generate random integer between $high$ and low. N is the number of layers in the given DNN model ($N = 5$ for LeNet). Figure 2 (b) shows several possible initial candidates generated based on the upper and lower bound.

Authorized licensed use limited to: Georgia Institute of Technology. Downloaded on January 19,2023 at 15:20:33 UTC from IEEE Xplore. Restrictions apply.

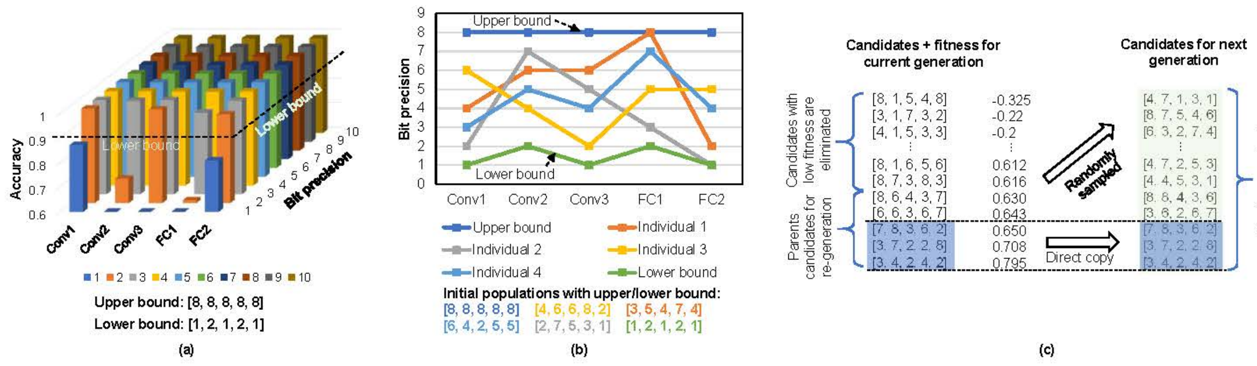


Fig. 2. (a) Layer sensitivity towards quantization for LeNet to determine the lower bound, (b) randomly generated initial candidates constrained by the upper bound and lower bound, and (c) procedure for re-generation of candidates.

Step ❷: Evaluation. The fitness function (F) evaluates two metrics of each candidate (C): the compression ratio and the computing accuracy. The fitness function is given by:

$$F(C) = -\alpha \cdot \sum_{i=0}^{N} C_i W_i - \beta \cdot Err$$
$$where \quad Err = \begin{cases} Err & acc \geq threshold \\ inf & acc < threshold \end{cases} \tag{2}$$

where C is the candidate solution (i.e. a vector with N integers); C_i is the bitwidth of i-th layer of the DNN and W_i is the number of weight parameters in that layer; Err is the error introduced by quantization and a penalty (inf or ∞) is applied to the fitness function if the error exceeds a pre-defined threshold; α and β are weighting factors for compression rate and accuracy, respectively. It is worth to mention that we directly quantize a pre-trained model and perform inference, no re-training is required.

Step ❸: Re-generation (Crossover and mutation). Based on the fitness value, good candidates (i.e. candidates with high fitness) are reserved to reproduce the next generation populations and the bad candidates are eliminated. Each generation is composed of 15 independent candidates, ranked based on their fitness value (Figure 2 (c)). The bottom 5 candidates (candidates with best fitness) are used for re-generation. To generate a new child candidate, two parent candidates are randomly picked from the 5 good candidates. The child candidate is generated based on the range described by the two parent candidates, regulated by equation 3.

$$C = [randi(high = max_i, low = min_i) \ \ for \ \ i \ \ in \ [0...N]]$$
$$where \ \ max_i = max(A_i, B_i) + 1 \tag{3}$$
$$min_i = min(A_i, B_i) - 1$$

where A_i and B_i are the bitwidth of the i-th layer defined by the randomly picked parent candidates A and B, respectively. Essentially, the two parents candidates define the upper/lower bound to reproduce a new candidate. To ensure the best results and fast convergence, at each iteration, 3 candidates with highest fitness value from last generation is append to the newly generated individuals (i.e. direct copy).

Step ❹: Fine tuning after GA. While standalone GA works perfectly on shallow models such as LeNet and AlexNet, we observe that the output (layer-wise quantization strategy) from GA alone for deep models still has room to be further compressed. Therefore, we apply the greedy search (GS) algorithm to the output of GA to further reduce the bitwidth of layers that are robust toward precision reduction. GS is an iterative method but performs search in a 'greedy' approach. At each step, GS iteratively reduces 1-bit for a layer and checks which layer shows the best robustness towards bit reduction; Then the precision for the most robust layer will be reduced accordingly. The GAQ with GS shows 5%-10% higher compression (same accuracy) for VGG and ResNet.

Again, for all steps, a pre-trained model is quantized and only inference is performed, no re-training is required.

Quantization for activation. We observe that the quantization sensitivity for activation are closely correlated with the sensitivity of weight parameters and in most cases, tend to be less robust (i.e. larger accuracy drop when using the same precision of the weight in the same layer). We determine the precision for the activation using

$$P_{act}^{i} = min(P_u, P_{weg}^{i} * 2) \tag{4}$$

where, P_{act}^{i} and P_{weg}^{i} are the precision for activation and weight in ith layer, respectively. P_u is a dataset-dependent value which is used to constrain the maximum possible precision for activation. For MNIST and CIFAR-10, P_u is 8-bit; for ImageNet, P_u is 12-bit.

IV. Q-PIM HARDWARE ARCHITECTURE

Q-PIM architecture stems from the in-memory bit accumulation configuration but is enhanced to support dynamic precision. Figure 3 shows the Q-PIM architecture with three levels hierarchy: VMM (vector matrix multiplication) engine, PIM core, and system implementation.

VMM engine. VMM engine is used for vector-matrix multiplication and is the basic computing unit in the Q-PIM architecture. VMM engine is composed of one memory crossbar, WL/BL peripherals, input activation buffer, and a simple control logic. Note, PIM architecture is a weight stationary computing machine where the weight parameter is static while the activation changed dynamically.

Authorized licensed use limited to: Georgia Institute of Technology. Downloaded on January 19,2023 at 15:20:33 UTC from IEEE Xplore. Restrictions apply.

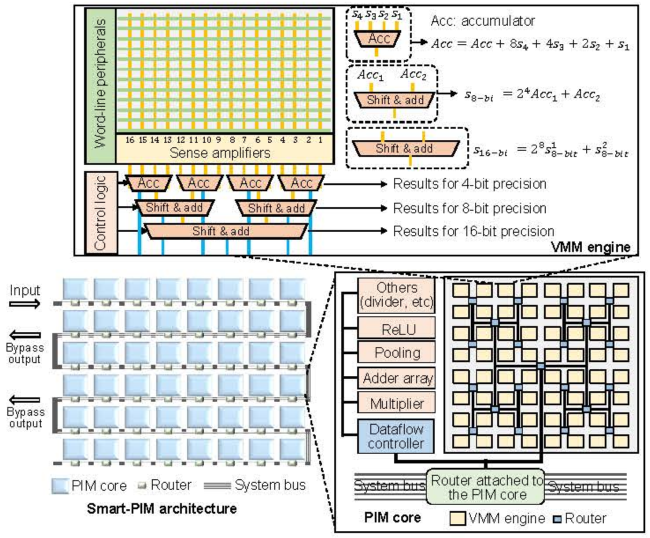


Fig. 3. The Q-PIM system architecture showing the hierarchy of VMM engine (top), PIM core (left), and top-level system (right) organization.

The flexible bit precision is achieved by a set of hierarchical organized *shifter & adder* units (top of Figure 3). Assuming we have a memory crossbar contains 16 BLs and each weight parameter is a 4-bit number (be programmed to 4 adjacent cells), each first level *accumulator* accepts results from 4 BLs. After accumulating results from all rows, the result can be directly sent to the output buffer via the bypass connections (blue lines). Alternatively, if the data precision is 8-bit, the results from two adjacent accumulators are routed to the first level shifter & adder which can perform the shifting and adding operation over the output from the above two accumulators (corresponding to 8 bitlines). Eventually, with a larger memory array and more hierarchical shifter & adder blocks, we can perform fixed point MAC operation with different bit precision in the same place. The design overhead for enabling flexible precision is discussed in Section V-B.

PIM core. PIM core is composed of multiple VMM engines placed in a 2-D plate, interconnected with a customized hierarchical network-on-chip (H-NoC) [10] to support flexible bit-precision. Additionally, function units supporting non-matrix operation such as element-wise multiplication and activation functions are implemented, ensuring the flexibility to support computations beyond matrix operation.

System architecture. At the top level, we implement Q-PIM system architecture where PIM cores are connected by the routing architecture. Q-PIM is an inference only accelerator and the data back-propagation is not considered. Therefore, there are three possible routing/mapping scenarios depending on the DNN layer size: (i) One layer is mapped to one PIM core. The router receives activation, dispatches data into PIM core for computing, and then sends the new activation to the next PIM core. (ii) Multiple small layers are mapped to one PIM core. (iii) A large layer is mapped to multiple consecutive PIM cores (only happens for fully-connected layer in our experiments).

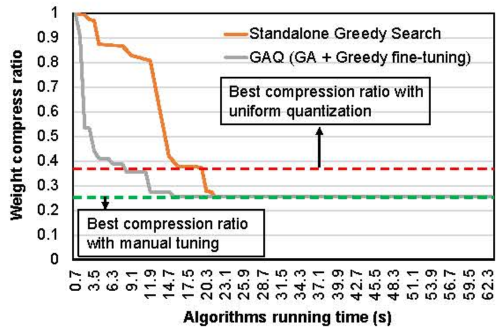
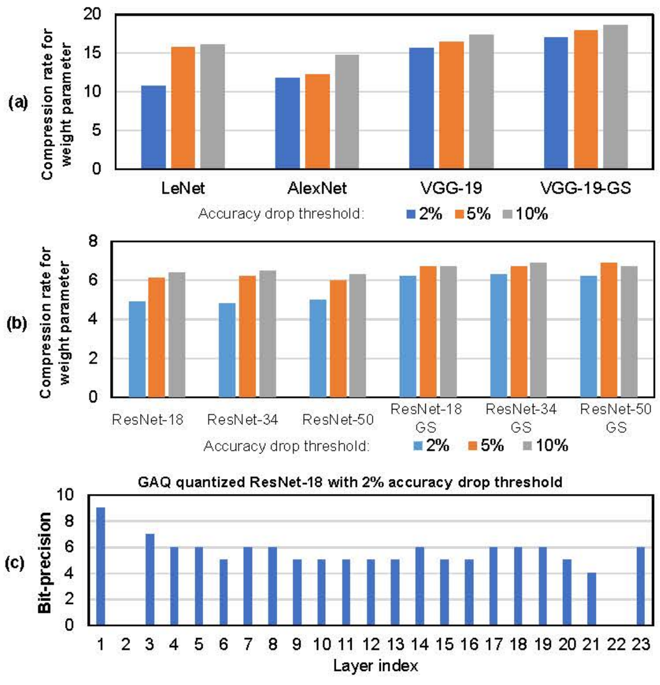


Fig. 4. Convergence behavior (achieved compression ratio) of GAQ showing as a function of running time. GAQ shows faster convergence than using a standalone Greedy Search. The red dash line is the compression ratio for the uniform quantization (3-bit) under $< 2\%$ accuracy drop constrain. The green dash is the optimal compression strategy by handcrafted tuning under the same constrain, [1 3 2 2 2] bits for the 5 layers of LeNet.



Parameter size/accuracy before and after compression (<2% accuracy drop)

	LeNet	AlexNet	VGG	ResNet-18	ResNet-34	ResNet-50
Dataset	MNIST	Cifar-10		ImageNet		
Floating-point parameter size	0.25 MB	32.0 MB	172 MB	46.7 MB	87.1 MB	102 MB
GA compressed	0.02 MB	2.78 MB	11.0 MB	7.58 MB	13.1 MB	17.6 MB
Float-point accuracy	98.5%	84.7%	91.7%	70.4%	73.8%	76.4%
Compressed accuracy	96.7%	82.9%	89.7%	68.9%	71.9%	74.6%

Fig. 5. Parameter compression rate for (a) LeNet, AlexNet, VGG-19 and (b) ResNet-18/34/50 under different accuracy drop constrains (2%, 5%, and 10%). (c) Layer-wise quantization for ResNet-18 with 2% accuracy drop constrain. Insert table shows the parameter size in MB before and after compression.

V. Simulation Results

A. GAQ algorithm evaluation

GAQ is implemented with Tensorflow. The benchmarks includes: LeNet for MNIST dataset, AlexNet/VGG for CIFAR-10 dataset, and ResNet-18/34/50 for ImageNet dataset.

Authorized licensed use limited to: Georgia Institute of Technology. Downloaded on January 19,2023 at 15:20:33 UTC from IEEE Xplore. Restrictions apply.

Convergence Behavior of GAQ. Figure 4 shows the convergence behavior of GAQ for LeNet on MNIST dataset. The GAQ shows more compression than a uniform quantization for same accuracy loss target, and achieves the same optimal compression obtained by manual tuning. We also compare GAQ with a standalone greedy search algorithm. The standalone GS shows decent compression rate even though the convergence is slower than GAQ. However, a standalone GS does not scale to deeper models as the computational complexity increase dramatically ($\mathcal{O}(N^2M)$ where N is the number of layers in DNN model and M represents the computation complexity in that model). For instance, with VGG, standalone GS take 4x longer time than GAQ to reach convergence while the compressed model is 15% larger. Overall, GAQ shows better compression rate and faster convergence ($\mathcal{O}(M)$).

Evaluation on MNIST and CIFAR-10. Figure 5 (a) shows the weight compression rate for LeNet, AlexNet and VGG under different accuracy drop constrain (2%, 5%, 10%). For VGG, GS based fine tuning is applied to bring another $\sim 10\%$ compression. With 2% accuracy drop constrain, the average compression is 13.7x across the DNN models and the number becomes 16.7x for 10% constrain.

Evaluation on ResNet with ImageNet Dataset. ResNet are much more sensitive to quantization. Without GS based fine tuning, the average weight compression is 4.9x with 2% accuracy drop constrain (Figure 5 (b)). The model size can be further reduced using GS (named as ResNet-xx-GS in Figure 5), resulting to 6.3x reduction. Figure 5 (c) shows the GAQ layer-wise quantization results for ResNet-18 with 2% accuracy drop constrain. The insert table in Figure 5 shows the compressed weight parameter size for the benchmark DNN models.

Comparison with other DNN quantization works. Table I shows a detailed comparison between the proposed algorithm and other state-of-the-art DNN quantization methods, including LQ-NETs [3], UNIQ [4], DoReFa [2], Apprentic [1], PACT [6], and AQN [5]. We only include works that contain results for ResNet on ImageNet dataset into our comparison. Except AQN, all other works rely on multi/one pass training. For Apprentice and PACT which achieve 2-bit weight precision (i.e.best weight compression rate), handcraft tuning is required to determine which layers are reserved in floating-point precision. Although, AQN is a fully automated approach but the compression rate is less compared to GAQ.

On average, GAQ reduces the weight parameter to $\sim$ 5-bit with insignificant accuracy drop ($< 2\%$). While our approach shows less compression than the 2-bit quantization algorithms such as Apprentice [1] and PACT [6], we argue the key advantage over other algorithms is that GAQ works without time consuming training nor large training dataset. Further, our algorithm is much flexible with no requirement of manual tuning for layer-wise quantization.

B. Q-PIM hardware evaluation

We implement an illustrative Q-PIM with 6-T SRAM in 28nm technology. We consider 256×256 memory crossbar and design the SRAM to operate at 4 GHz while the digital logic at 1 GHz, all verified using SPICE simulation. The inter and intra PIM core NoC bandwidth are set to be 256 GB/s. We develop design automation tool flow to automatically generate full-chip physical of Q-PIM (Figure 6). Additionally, the EDA flow performs design space exploration to generate a Q-PIM architecture given a target on-chip memory capacity.

Our illustrative Q-PIM design contains 20 PIM cores with 20MB on-chip memory capacity. The physical design based power, timing, and area are used to drive our analysis. Table II illustrates the detailed system implementation and throughput/efficiency under different precision. The Q-PIM

TABLE I
COMPARISON AGAINST OTHER SOTA DNN QUANTIZATION WORKS WITH FOCUSES ON RESNET AND IMAGENET DATASET.

ResNet-18:

Algorithms	Quantization	Training complexity	Top-1 floating	Top-1 quantized	Degradation
LQ-NETs	W4/A32	Multi-pass	70.3	70.0	0.3
PACT	W2/A2 and floating point	One-pass	70.4	67.0	3.4
DoReFa	Binary	One-pass	70.2	62.6	7.6
Our work	**Layer-wise Avg. 5.1-bit**	**No need**	**70.4**	**68.9**	**1.8**

ResNet-34:

Algorithms	Quantization	Training complexity	Top-1 floating	Top-1 quantized	Degradation
Apprentice	W2/A8 and floating point	Multi-pass	73.6	71.5	2.1
Our work	**Layer-wise Avg. 5.1-bit**	**No need**	**73.8**	**71.9**	**1.9**

ResNet-50:

Algorithms	Quantization	Training complexity	Top-1 floating	Top-1 quantized	Degradation
UNIQ	W4/A8	One-pass	76.0	73.4	2.6
PACT	W2/A2 and floating point	One-pass	76.9	74.2	2.7
AQN*	Layer-wise Avg. ~ 6-bit	No need	76.4	74.8	1.2
Our work	**Layer-wise Avg. 5.2-bit**	**No need**	**76.4**	**74.6**	**1.8**

*Results for AQN is projected from the figures in [5].

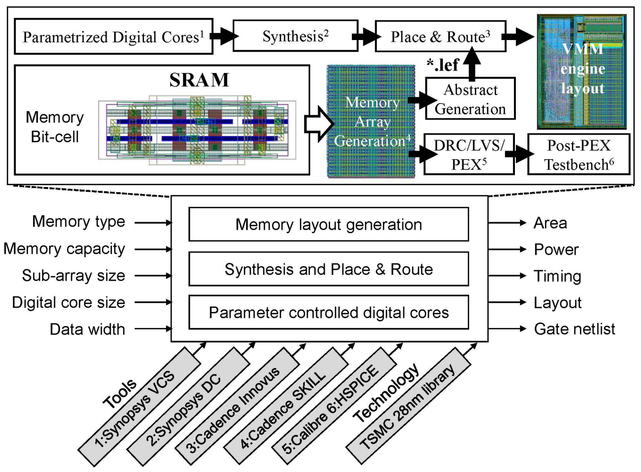
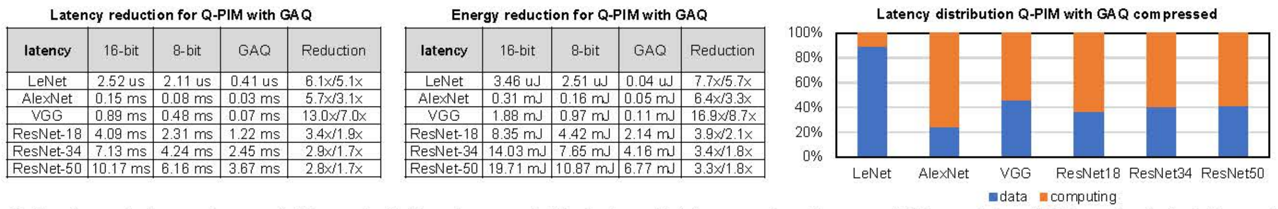


Fig. 6. The design automation flow for generating Q-PIM.

Authorized licensed use limited to: Georgia Institute of Technology. Downloaded on January 19,2023 at 15:20:33 UTC from IEEE Xplore. Restrictions apply.



Latency reduction for Q-PIM with GAQ

latency	16-bit	8-bit	GAQ	Reduction
LeNet	2.52 us	2.11 us	0.41 us	6.1x/5.1x
AlexNet	0.15 ms	0.08 ms	0.03 ms	5.7x/3.1x
VGG	0.89 ms	0.48 ms	0.07 ms	13.0x/7.0x
ResNet-18	4.09 ms	2.31 ms	1.22 ms	3.4x/1.9x
ResNet-34	7.13 ms	4.24 ms	2.45 ms	2.9x/1.7x
ResNet-50	10.17 ms	6.16 ms	3.67 ms	2.8x/1.7x

Energy reduction for Q-PIM with GAQ

latency	16-bit	8-bit	GAQ	Reduction
LeNet	3.46 uJ	2.51 uJ	0.04 uJ	7.7x/5.7x
AlexNet	0.31 mJ	0.16 mJ	0.05 mJ	6.4x/3.3x
VGG	1.88 mJ	0.97 mJ	0.11 mJ	16.9x/8.7x
ResNet-18	8.35 mJ	4.42 mJ	2.14 mJ	3.9x/2.1x
ResNet-34	14.03 mJ	7.65 mJ	4.16 mJ	3.4x/1.8x
ResNet-50	19.71 mJ	10.87 mJ	6.77 mJ	3.3x/1.8x

Fig. 7. Per image inference latency (table on the left) and energy (table in the middle) comparison between 16-bit precision (16-bit activation), 8-bit precision (16-bit activation), and GAQ compressed DNN models running on Q-PIM hardware. The sub figure on the right shows the latency distribution.

TABLE II
Q-PIM IMPLEMENTATION DETAILS.

SRAM based Q-PIM implementation

Memory solution	PIM core Power (W)	PIM core Area (mm²)	Memory Capacity	Peak Throughput (8-bit precision)
SRAM	3.15 W	7.65 mm²	20 MB (20 PIM cores)	41 TOPs (Clk_mem: 4 GHz)

System throughput at different precision

	4-bit	8-bit	16-bit	32-bit
Throughput	163.8 TOPs	41.0 TOPs	10.2 TOPs	2.6 TOPs
Density	1072 GOPs/mm²	268 GOPs/mm²	66.8 GOPs/mm²	17.0 GOPs/mm²
Efficiency	2588 GOPs/W	649 GOPs/W	161 GOPs/W	41 GOPs/W

TABLE III
COMPARISON WITH OTHER DNN ACCELERATORS.

	Hardware	Training support	Precision	Parameter storage	Computing Efficiency
DaDianNao	ASIC	No	16/32-bit	eDRAM	286 GOPs/W
TPU-v2	ASIC	Yes	8-bit, 16-bit bfloat	DRAM	180 GOPs/W
Deep train	NMP*	Yes	8-bit, 32-bit float	DRAM	566 GOPs/W
ISAAC	PIM	No	16-bit	ReRAM	381 GOPs/W
Neural Cache	PIM	No	Arbitrary	SRAM	529 GOPs/W
FERA	PIM	No	8-bit	FeFET	443 GOPs/W
Q-PIM (Ours)	**PIM**	**No**	**4/8/16/32-bit**	**SRAM**	**649 GOPs/W**

*NMP: Near memory processing

demonstrates the state-of-the art performance (under 8-bit precision) as shown in Table III.

Overhead for flexible precision. Assuming we perform 8-bit fixed point operation using a 256×256 memory array, it takes $256 \times 8 = 2048$ cycles to accumulate data (i.e. the 256 rows are iterated 8 times). Another 3 clock cycles are required for the data going through the shifter & adder hierarchy. The overhead due to flexible bit-precision is trivial in terms of latency ($< 1\%$) and power (7%). However, our design introduces large area overhead (47%) due to the registers for intermediate data storage inside shifter & adder.

System performance with GAQ. Without any re-training or fine tuning, directly quantizing a pre-trained model parameters to 16-bit yields the same accuracy of floating point precision for all benchmark DNN models. However, further reducing weight parameter to 8-bit introduces large accuracy drop ($> 10\%$ for ResNet). Figure 7 shows the per image inference latency and energy for benchmark models with 16-bit, 8-bit, and GAQ compressed precision in Q-PIM platform. With the same hardware, GAQ compressed models demonstrate $2.8\times$ to $13\times$ latency reduction and $3.3\times$ to $16.9\times$ energy reduction over the 16-bit precision with only insignificant accuracy loss, respectively.

It is worth to mention that while the GAQ compressed models can have arbitrarily bit precision, Q-PIM hardware only supports 4/8/16/32-bit precision, therefore, we round-up the precision to make sure there is no additional accuracy loss.

VI. CONCLUSION

This paper presents a genetic algorithm (GA) based training free layer-wise quantization method, named as GAQ, to reduce model complexity of arbitrary DNN architectures. The paper demonstrates GAQ on a SRAM based flexible-precision all-digital PIM architecture (Q-PIM). The simulation in 28nm CMOS shows potential for significant energy and latency advantage using GAQ and Q-PIM over fixed-precision PIM architectures. As the DNN model complexity continues to grow, Q-PIM will be an attractive candidate for energy-efficient acceleration of deep learning. Integration of Q-PIM with pruning and knowledge distillation for further compression or better accuracy are important future works.

REFERENCES

[1] Asit Mishra and Debbie Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852*, 2017.
[2] Shuchang Zhou et al. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.
[3] Dongqing Zhang et al. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 365–382, 2018.
[4] Chaim Baskin et al. Uniq: Uniform noise injection for non-uniform quantization of neural networks. *arXiv preprint arXiv:1804.10969*, 2018.
[5] Yiren Zhou et al. Adaptive quantization for deep neural network. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
[6] Jungwook Choi et al. Accurate and efficient 2-bit quantized neural networks. In *SysML 2019*, 2019.
[7] Song Han et al. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.
[8] Ali Shafiee et al. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News*, 44(3):14–26, 2016.
[9] Charles Eckert et al. Neural cache: Bit-serial in-cache acceleration of deep neural networks. *arXiv preprint arXiv:1805.03718*, 2018.
[10] Yun Long et al. A ferroelectric fet based processing-in-memory architecture for dnn acceleration. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 2019.
[11] Shuangchen Li et al. Drisa: A dram-based reconfigurable in-situ accelerator. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 288–301. IEEE, 2017.
[12] Mohsen Imani et al. Floatpim: In-memory acceleration of deep neural network training with high precision. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 802–815. ACM, 2019.

Authorized licensed use limited to: Georgia Institute of Technology. Downloaded on January 19,2023 at 15:20:33 UTC from IEEE Xplore. Restrictions apply.