Flexible Low Power CNN Accelerator for Edge Computing with Weight Tuning

Flexible Low Power CNN Accelerator for Edge Computing with Weight Tuning

Miaorong Wang, Anantha P. Chandrakasan
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139
Email: {miaorong, anantha}@mit.edu

***Abstract*—To support various edge applications, a neural network accelerator needs to achieve high flexibility and classification accuracy within a limited power budget. This paper proposes a weight tuning algorithm to improve the energy efficiency by lowering the switching activity. A flexible and runtime-reconfigurable CNN accelerator is co-designed with the algorithm and demonstrated with a feature extraction processor on an FPGA. The system is fully self-contained for small CNNs and speech keyword spotting is shown as an example. A fully integrated custom ASIC is also being fabricated for this system. Based on post place-and-route simulation of the ASIC, the weight tuning algorithm reduces the energy consumption of weight delivery and computation by 1.70x and 1.20x respectively with little loss in accuracy.**

I. Introduction

Smart edge devices that support efficient neural network (NN) processing have recently gained public attention. With algorithm development, previous work has proposed small-footprint NNs achieving high performance in various medium complexity tasks, e.g. speech keyword spotting (KWS), human activity recognition (HAR), etc. Among them, convolutional NNs (CNNs) perform well [1], which gives rise to the deployment of CNNs on edge devices. A hardware platform for edge devices should be (1) flexible to support various NN structures optimized for different applications; (2) energy efficient to operate within the power budget; (3) achieving high accuracy to minimize spurious triggering of power-hungry downstream processing, since it is often part of a large system.

Both algorithms, such as quantization and model compression, and accelerator designs for energy efficient processing of CNNs have been proposed. Quantization reduces the bit precision. But some experiments show that quantizing NNs to extremely low bitwidth, e.g. 1 bit, does not necessarily lead to model size reduction, because the model structure needs to be modified to retain the accuracy [2]. Model compression algorithms focus on minimizing the model size with little loss in accuracy. However, pruning-based algorithms mainly need specialized hardware to exploit the resulting sparse tensors for energy reduction. Previous work has proposed CNN accelerators targeting edge computing. However, many of them support limited flexibility for the NN shapes, are designed only for a specific task or sacrifice the accuracy [3]–[5].

To address the challenges in flexibility, energy efficiency and accuracy in CNN accelerator design, this work takes an algorithm-and-hardware co-design approach. The key contributions of this paper are highlighted as follows: (1) a weight tuning algorithm that reduces the energy consumption associated with weight delivery and computation by lowering the toggle count of weight sequence; (2) the co-design of a CNN accelerator that supports the proposed algorithm and is flexible for a wide range of NN model structures; (3) the demonstration of speech keyword spotting (KWS) as an example on an FPGA and the design of a fully integrated ASIC (being fabricated) with the proposed CNN accelerator and a feature extraction processor[1].

II. Weight Tuning Algorithm

The weight tuning algorithm reduces the energy consumption of the CNN accelerator with little loss in accuracy by tuning the bit representation of weights. Fundamentally different from quantization and model compression algorithms, the proposed algorithm is based on the theory that the dynamic power of a CMOS gate is linearly proportional to its switching activity, which is influenced by the toggle count (TC) of its input sequence. In a NN accelerator, weights are read from the memory, delivered through network-on-chip (NoC) and then multiplied with input activations (IAs) following a sequence set by the designer. The TC of this weight sequence affects the switching activity of weight buses and the multipliers. And for a memory that can do conditional pre-charge based on previously read data, e.g. [7], it also affects the pre-charge activity of bit-lines. Therefore, minimizing the TC of weight sequence can reduce the power consumption of those components. To gain those benefits, we propose a weight tuning algorithm that contains 3 sequential steps: (1) tensor decomposition with retraining; (2) quantization and sign-magnitude representation; (3) weight scaling and bit perturbation with retraining.

A. Tensor Decomposition (TD) with Retraining

TD with retraining [8] is a model compression method that breaks a convolutional layer into 3 layers without activation function in between and retrains to maintain accuracy as shown in Fig. 1. The total parameters and computation of the resulting layers are less than those of the original layer. Thus it is favorable for reducing the energy per inference.

[1]This unit was first designed by M. Price [6] and then modified by S. Lauwereins from Prof. Marian Verhelst's group at MICAS – KU Leuven.

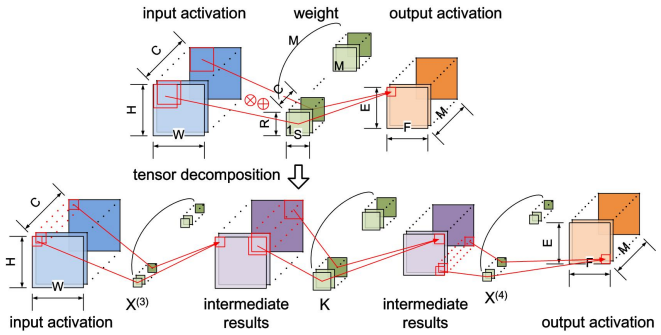


Fig. 1. An illustration of TD for CNN [8] and the shape parameters used in this paper. The decomposed NN is then retrained as proposed in [8].

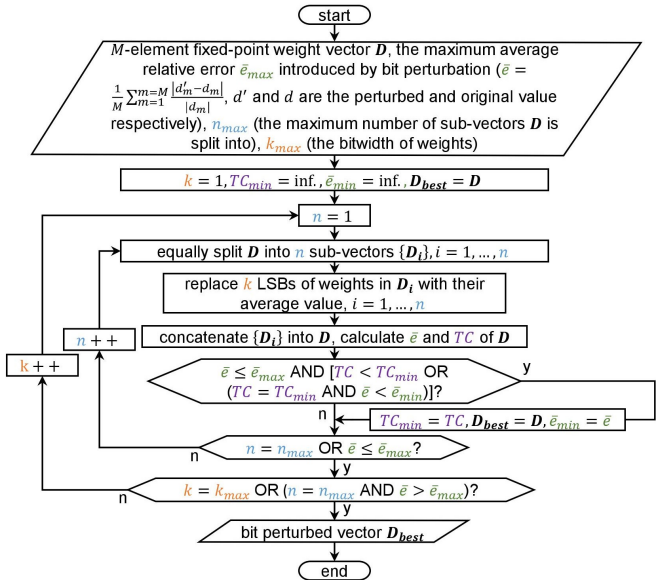


Fig. 2. The flowchart of the bit perturbation algorithm. Relative error $e = \frac{|d' - d|}{|d|}$, where d' and d is the perturbed and original value respectively, of weight is used to evaluate the deviation caused by bit perturbation.

B. Quantization and Sign-Magnitude (QSM) Representation

We then convert the model to 8 bit fixed-point numbers with linear quantization as used in many NN accelerators. Instead of 2's complement (2C) format, we use sign-magnitude (SM) representation of weights to reduce weight sequence TC [9], [10]. The overhead of that is the implementation of adder. Section III discusses in detail how we minimize the overhead.

C. Weight Scaling and Bit Perturbation (WSBP) with Retraining

The 4-D weight tensor of every layer is flattened to a 1-D vector following the sequence that weights are read, delivered and calculated in the NN accelerator. We sequentially apply weight scaling and bit perturbation to it to further reduce the TC and incorporate them with retraining to maintain accuracy.

1) Weight Scaling: Weight Scaling is to scale the weights and biases uniformly in every layer to lower the TC of weight sequence, which is inspired by the coefficient scaling for FIR filters [11]. The scaling factor K_l of layer l is determined by $K_l = argmin_{k_l} \texttt{ToggleCount}(k_l \mathbf{W_l}), k_l \in (a, b), a \ge 0$, where $\mathbf{W_l}$ is the weight tensor, ToggleCount is the function to calculate the TC and a, b are the user defined bounds of K_l. K_l is determined by an exhaustive search in the given range with a predefined step size s. It can be applied to layers using ReLU as the activation function without impact on the classification accuracy, given that $ReLU(K_l \mathbf{x}_l) = K_l ReLU(\mathbf{x}_l)$.

2) Bit Perturbation: Inspired by the coefficient perturbation for FIR filters proposed in [11], we apply the bit perturbation to weight sequence as shown in Fig. 2 to reduce the TC. It equally splits the weight vector $\mathbf{D}$ into n sub-vectors $\{\mathbf{D_i}\}$ and then replaces k LSBs of weights in every sub-vector $\mathbf{D_i}$ with their average value. It loops through all possible n and k searching for a bit tuned low-TC weight sequence with small deviation from the original weights.

3) Retraining: Retraining is applied to restore the accuracy loss caused by WSBP. The proposed QSM and WSBP are applied to the pretrained floating point NN as a wrapper function of parameters in the forward pass. During back-propagation, the straight-through estimator (STE) [12] is adopted, which passes the gradients through the wrapper function as-is.

III. System Architecture

As shown in Fig. 3, we implemented a standalone system with 80kB model parameter memory and 1kB configuration buffer for storage and configuration of the entire NN with up to 12 layers during the setup phase. All the data buffering is done on-chip using a 2kB circular input buffer and a 48kB activation memory without the need of off-chip DRAM.

The processing element (PE) array level dataflow is shown in line 4 – 8 of Fig. 4. The PE array is logically treated as having $C1$ columns and $M1$ rows as shown in line 7–8. Reconfigurable NoCs are used to support that flexibility with NoC controller adopted from Eyeriss [13]. Following this logical mapping, weights are unicast to PEs, and thus tree-structured NoC with a depth of 2 as shown in Fig. 3 is used for weight delivery. The ID of every controller at each level is unique and fixed. Activations are multicast across multiple logical rows. To support that, only level 0 NoC controllers are used. Their IDs, determined by $M1s$ and $C1$, are configured at runtime before the execution of every layer. The delivery of bias, partial sum and output activations are similar except that not every PE needs data. Thus the unused controllers and FIFOs can be gated. Different from Eyeriss which implements row stationary (RS) dataflow, the proposed design follows weight stationary (WS) dataflow in the PE array level considering that 1×1 or $1 \times x$ convolutional layers resulting from TD take up a large part of the CNNs. Fully-connected layers can be organized as a special case of convolutional layers where $W = R, H = S, E = 1, F = 1$

The PE level dataflow is shown in line 9 – 10 of Fig. 4 and the PE structure is illustrated in Fig. 5. PE has local storage to hold $M0 \times C0$ weights and $C0$ IAs to achieve temporal reuse of $E \times F$ and $M0$ times respectively as shown in line 6 and 9 of Fig. 4. $M0$ and $C0$ are runtime configurable to balance the

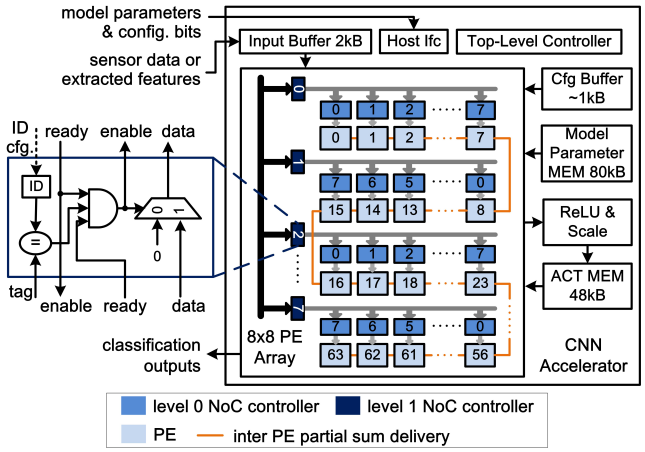


Fig. 3. System architecture of the NN accelerator and the micro-architecture of the NoC controller.

```
1)    int i[C][W][H];      // Input activations, C = C1*C0 <= 256
2)    int w[M][C][R][S];   // Filter weights, M = M1t*M1s*M0 <= 256
3)    int o[M][E][F];      // Output activations, E*F*M <= 48k

      // PE array level -- temporal
4)    for (r1=0; r1<R; r1++) { for (s1=0; s1<S; s1++) {
5)      for (m1t=0; m1t<M1t; m1t++) {  // R*S*M1t <= 2^16
6)        for (e1=0; e1<E; e1++) { for (f1=0; f1<F; f1++) {
            // PE array level -- spatial
7)          parallel_for (c1=0; c1<C1; c1++) {
8)          parallel_for (m1s=0; m1s<M1s; m1s++) {  // C1 * M1s <= 64
              // PE level -- temporal
9)            for (m0=0; m0<M0; m0++) {  // M0 = {0, 1, 2, 3}
10)             for (c0=0; c0<C0; c0++) {  // C0 = {3, 4}
11)               c = c0 + c1*C0;
12)               m = m0 + m1s*M0 + m1t*M0M1s;
13)               r = r1; s = s1; e = e1; f = f1;
14)               w = U*e1 + r; h = V*f1 + s;  // U, V: filter stride in width and height
15)               o[m][e][f] += i[c][w][h] * w[m][c][r][s]; // U,V <= 1024
      } ... }
```

Fig. 4. The dataflow illustrated by loop nests. The tensor dimension parameters are illustrated in Fig. 1. Bias is ignored for simplicity and batch size = 1 for real-time application. For the first layer, where $C = 1$, we replace C with R and remove the original R loop to increase the PE array utilization. The read, delivery and computation sequence of weight is as shown. PEs are filled up with weights in sequence. The limits on supported NN shapes are annotated.

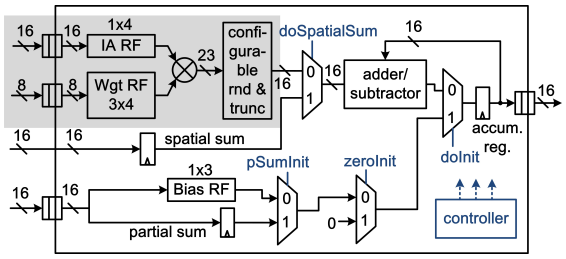


Fig. 5. The PE structure. The shaded part is the SM domain. The rest is in the 2C domain.

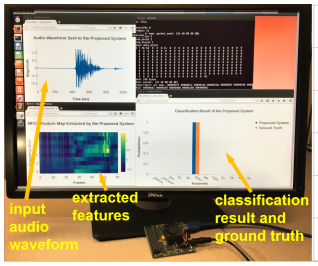
workload between PEs given layer shapes. All PEs are chained in a sequence for the inter-PE partial sum delivery as shown in Fig. 3. That supports spatial sum with flexible configuration of logical rows and columns, since partial sums can be spatially accumulated across an arbitrary number of PEs.

As discussed in Section II-B, weights are represented in SM format to reduce TC. Exploiting the fixed calculation pattern in CNN, we implement a mixed-representation datapath. Weights and IAs are multiplied in SM format using an unsigned multiplier and an XOR gate that generates the sign bit. An adder-subtractor is used to convert the SM product to 2C representation with the sign bit of the product as the carry bit, and do accumulation. The 2C outputs are delivered to other PEs for spatial sum or buffered in memory for temporal accumulation to generate the output activations. Only after all the computation of this layer is finished, do we need to convert them back to SM format for the next layer. Thus the energy overhead of the conversion is mitigated.

IV. Implementation Results

The CNN accelerator with a feature extraction processor is implemented on Xilinx XC7K410T. The CNN accelerator operates at 50MHz and consumes 68mW based on Vivado power estimation. The FPGA demo for the KWS task and the post place-and-route (P&R) resource utilization is shown in Fig. 6.

The proposed system is also being fabricated using TSMC 40nm LP process with a core area of 2.16 mm^2. Based on the ASIC design, we evaluate the weight tuning algorithm on several CNNs designed for KWS on speech command dataset [14], including CNNs under 80kB (referred to as CNN80) and 200kB memory constraints in [1] and the fstride-4 model in [15]. TD with retraining is applied to most layers except ones that largely impact the accuracy, e.g. the last layer. The resulting models in 2C format serve as the baseline. QSM and WSBP with retraining is applied to the decomposed layers with a step size s of 0.05, the scaling factor bounds $a = 0.8, b = 1.8$, $e_{max} = 0.15$ and n_{max} equal to half of the vector length. It reduces weight sequence TC by 1.79x–2.56x with less than 0.75% accuracy loss on the testing set for those cases. As shown in Fig. 7, the Hamming distance between successive weights are reduced. To analyze energy reduction, we implement a baseline CNN accelerator that uses 2C MACs following the same procedures for synthesis and P&R as the proposed accelerator using the same technology. Parasitics, SDF and SAIF obtained after P&R are annotated during power analysis. We simulate the execution of a tensor decomposed CNN80 on the dataset. Table I summarizes the total energy consumption of different components during the execution, and compares the accuracy and TC. $E_{mult.}$ and $E_{add.}$ are the total energy consumption of all the multipliers and adders in the PEs respectively. E_{MAC} is the sum of them. E_{wgtBus} is the energy of weight buses between the weight memory and PEs. It is obtained by summing up the internal and switching energy of buffers inserted in between. As shown, the weight tuning algorithm with the mixed-representation MAC reduces the computation energy by 1.20x compared to the 2C baseline. The energy of weight buses is reduced by 1.70x. Although the energy consumption of memory and activation delivery



	Availa-ble	Whole System	CNN accel.
LUT	254200	74059 (29%)	62473 (25%)
LUT-RAM	90600	5767 (6%)	5718 (6%)
FF	508400	18558 (4%)	17608 (3%)
BRAM	795	81 (10%)	33 (4%)
DSP	1540	142 (9%)	64 (4%)

Fig. 6. The FPGA demo of the CNN accelerator with a feature extraction processor on KWS, and a summary of FPGA post-P&R resource utilization.

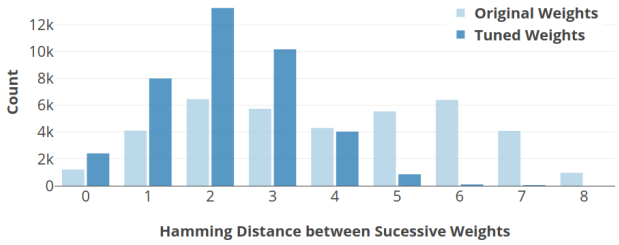


Fig. 7. The weight tuning algorithm reshapes the distribution of Hamming distance between weights with tensor-decomposed CNN80 as an example.

TABLE I
THE EFFECT OF THE WEIGHT TUNING ALGORITHM ON ACCURACY AND ENERGY CONSUMPTION BASED ON POST-P&R SIMULATION

	original 2C	bit tuned SM	loss/reduction
acc. (%)	89.3	88.8	0.5%
toggle count	154k	86k	1.79x
$E_{mult.}$	1.58uJ	1.11uJ	1.42x
$E_{add.}$	0.55uJ	0.66uJ	0.83x
E_{MAC}	2.13uJ	1.77uJ	1.20x
E_{wgtBus}	96.91nJ	56.89nJ	1.70x
$E_{totalSwitch}$	8.94uJ	7.68uJ	1.16x

is not affected by the algorithm, 1.16x reduction in the total switching energy of the entire system $E_{totalSwitch}$ is observed.

Compared with other digital ASICs for KWS, e.g. [3]–[5], our design is flexible, accurate and achieves comparable energy efficiency. The proposed architecture supports flexible shape and stride of inputs and weights for up to 12 layers. However, [5] is restricted to a fixed structure, [3] supports up to 2 layers with up to 64 nodes/layer for LSTMs, and [4] is designed for a fixed input stride and 3x3 1-bit convolution. Our supported CNNs can achieve 91.6% accuracy with 12 output classes on the public available dataset [1], while [3]–[5] only report accuracy on custom designed dataset and [3] only shows binary classification. Based on the post-P&R power analysis, the proposed design achieves 24.38 pJ/MAC with 8bit weights and 16bit activations at the worst case corner of 0.99V with a latency of 10ms for real-time KWS processing.

V. CONCLUSION

We co-designed a weight tuning algorithm and a CNN accelerator to improve energy efficiency with little loss in accuracy. Furthermore, the accelerator features high flexibility and runtime reconfigurability to support various applications. It is demonstrated on an FPGA with a feature extraction processor for KWS. Moreover, a fully integrated ASIC is being fabricated. The post-P&R power analysis of the ASIC shows the proposed algorithm reduces the energy consumption of weight delivery and computation by 1.70x and 1.20x respectively. For future work, the weight tuning algorithm can be applied to other NNs, such as LSTM, co-designed with other hardware architectures and dataflows, e.g. output stationary dataflow, and exploited by data-dependent memory [7].

ACKNOWLEDGMENT

The authors would like to thank Foxconn Technology Group for supporting this project and the TSMC University Shuttle Plan for chip fabrication.

REFERENCES

[1] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello Edge: Keyword spotting on microcontrollers," *arXiv:1711.07128 [cs, eess]*, Nov. 2017. [Online]. Available: http://arxiv.org/abs/1711.07128

[2] G. Gobieski, B. Lucia, and N. Beckmann, "Intelligence Beyond the Edge: Inference on intermittent embedded systems," in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2019, pp. 199–213.

[3] J. Giraldo and M. Verhelst, "Laika: A 5uW programmable LSTM accelerator for always-on keyword spotting in 65nm CMOS," in *Proceedings of IEEE European Solid State Circuits Conference (ESSCIRC)*, 2018, pp. 166–169.

[4] S. Yin, P. Ouyang, S. Zheng, D. Song, X. Li, L. Liu, and S. Wei, "A 141 uW, 2.46 pJ/neuron binarized convolutional neural network based self-learning speech recognition processor in 28nm CMOS," in *IEEE Symposium on VLSI Circuits*, 2018, pp. 139–140.

[5] M. Shah, J. Wang, D. Blaauw, D. Sylvester, H.-S. Kim, and C. Chakrabarti, "A fixed-point neural network for keyword detection on resource constrained hardware," in *IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2015, pp. 1–6.

[6] M. Price, J. Glass, and A. P. Chandrakasan, "14.4 A scalable speech recognizer with deep-neural-network acoustic models and voice-activated power gating," in *IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, 2017, pp. 244–245.

[7] C. Duan, A. Gotterba, M. E. Sinangil, and A. P. Chandrakasan, "Reconfigurable, conditional pre-charge SRAM: Lowering read power by leveraging data statistics," in *Proceedings of IEEE Asian Solid-State Circuits Conference (A-SSCC)*, Nov. 2016, pp. 177–180.

[8] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *arXiv:1511.06530 [cs]*, Nov. 2015. [Online]. Available: http://arxiv.org/abs/1511.06530

[9] A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proceedings of the IEEE*, vol. 83, no. 4, pp. 498–523, 1995.

[10] P. N. Whatmough, S. K. Lee, D. Brooks, and G.-Y. Wei, "DNN engine: A 28-nm timing-error tolerant sparse deep neural network processor for iot applications," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 9, pp. 2722–2731, 2018.

[11] N. Kasturi, "Power reducing algorithms in FIR filters," Master's thesis, Massachusetts Institute of Technology, 1997.

[12] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," in *Proceedings of International Conference on Learning Representations (ICLR)*, 2016.

[13] 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, Jan. 2017.

[14] P. Warden. (2017) Speech command: A public dataset for single-word speech recognition. [Online]. Available: download.tensorflow.org/data/speech_commands_v0.01.tar.gz

[15] T. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Proceedings of Interspeech*, 2015.