BloomCA: A Memory Efficient Reservoir Computing Hardware Implementation Using Cellular Automata and Ensemble Bloom Filter

Dehua Liang
Osaka University
Osaka, Japan
d-liang@ist.osaka-u.ac.jp

Masanori Hashimoto
Osaka University
Osaka, Japan
hasimoto@ist.osaka-u.ac.jp

Hiromitsu Awano
Kyoto University
Kyoto, Japan
awano@i.kyoto-u.ac.jp

2021 Design, Automation & Test in Europe Conference & Exhibition (DATE) | 978-3-9819263-5-4/21/$31.00 ©2021110.23919/DATE51398.2021.9474047

***Abstract*—In this work, we propose a *BloomCA* which utilizes cellular automata (CA) and ensemble Bloom filter to organize an RC system by using only binary operations, which is suitable for hardware implementation. The rich pattern dynamics created by CA can map the input into high-dimensional space and provide more features for the classifier. Utilizing the ensemble Bloom filter as the classifier, the features can be memorized effectively. Our experiment reveals that applying the ensemble mechanism to Bloom filter endues a significant reduction in inference memory cost. Comparing with the state-of-the-art reference, the *BloomCA* achieves a 43× reduction for memory cost without hurting the accuracy. Our hardware implementation also demonstrates that *BloomCA* achieves over 21× and 43.64% reduction in area and power, respectively.**

***Index Terms*—BloomCA, Reservoir Computing, Cellular Automata, Ensemble Bloom Filter.**

I. Introduction

With the scale of deep neural networks (DNN) increasing, large computing demands are becoming unbearable for most wearable devices. Reservoir computing (RC) is a promising alternative to drastically reduce the computational burden of machine learning. The most critical advantage of RC is that only a part of the parameters are trained while the rest of them can be fixed. Owing to this unique feature, RC can be implemented with limited hardware resources.

The RC architecture generally consists of a *reservoir* and a *classifier*. All the input signals are given to a reservoir, which is often constructed by a recurrent neural network (RNN) whose synaptic weights are randomly initialized [1]. After being fed into the fixed and non-linear pattern dynamic reservoir, these input signals are mapped into higher dimensional feature space. Finally, the output is obtained by using a classifier whose parameters can be trained. Such a design strategy effectively avoids the complex training process in the reservoir. Because of its cheap computational cost in both the training and inference phase, the RC system has been successfully applied in many fields, such as image recognition or robot control [2]. However, frequent use of floating-point (FP) calculations makes the RC implementation on hardware challenges.

To reduce the massive usage of arithmetic units, cellular automata (CA) has been proposed as a promising alternative to realize reservoir with limited hardware resources [3]. CA consists of multiple cells aligned in a one-dimensional array, where each cell takes two possible discrete states and evolves in discrete time steps. With the rich pattern dynamic characteristic, CA is greatly suited to reservoir structure. A comprehensive study on utilizing CA as a reservoir is performed in [4], which achieves competitive accuracy performance. However, this model exploits softmax-function as their classifier, which still requires FP calculations, and hence it is not suitable for resource constrained devices.

Meanwhile, *Bloom WiSARD* has been proposed to implement pattern recognition systems. As a weightless neural network utilizing Bloom filters, this approach successfully eliminated FP calculation in both the training and inference phase [5]. Bloom filter is a data structure for approximate member query [6]. As a classical implementation of hyperdimensional (HD) computing, the Bloom filter is characterized by its simplicity both in software and hardware. Although [5] demonstrated that the Bloom filter has practically useful performance on image recognition tasks, a large amount of memory requirement is remaining as one of the bottlenecks, which makes its adoption impracticable when it comes to portable devices.

To further reduce the memory footprint, we propose a novel RC model: *BloomCA*. The uniqueness of *BloomCA* is to utilize the ensemble Bloom filter as a classifier, which contributes to the significant reduction of the memory cost. The main contributions are the followings:

- Eliminating FP calculations and integer multiplications.
- Achieving 43× memory reduction during inference in comparison with [5] without hurting the accuracy.
- Over 21× and 43.64% reduction in area and power consumption, respectively.

II. Preliminary

A. Reservoir Computing

Reservoir computing (RC) is a way of circumventing the difficulty in learning a number of RNN parameters, which has been successfully applied in numerous fields such as robot control [2] and image processing [4]. More recently, Echo state networks (ESN) [7] and liquid state machines (LSM) [8] have been proposed in different research domains. They are collectively referred to RC because both of them have the component *reservoir* [9]. The fixed and non-linear hidden layers are generally referred as a *reservoir*, and the output layer is considered as a *classifier*.

Comparing with the conventional DNN, RC systems use fixed weights in the hidden layers and only modify the weights of the output layer in the training process [1]. Since most of the weights are fixed, they can be implemented by using hardwired logic and thus do not require memory circuits. Hence, RC is considered to be a promising alternative to replace DNN whose model size continues to increase exponentially [10]. The state of each node is updated during M steps according to a non-linear mapping. However, such mapping still requires FP computations, which may lead to the constraint when considering the implementation onto portable devices. To eliminate FP computations, several works have proposed to exploit cellular automata (CA) as an alternative to traditional reservoirs.

B. Cellular Automata (CA)

CA is a simple dynamical system with a discrete state that evolves in discrete time steps. The evolution is determined by a simple rule based on the state of the neighboring cells and themselves. Depending on the specific rule, CA exhibits a wide variety of behaviors. Such architecture can not only provide rich pattern dynamics but also map the inputs into high dimensional feature space. CA has been proved to be useful both as a general model of complexity and as more specific representations of non-linear dynamics in different scientific fields [4].

Elementary cellular automata (ECA) is the simplest class of 1-dimensional CA [4], where each cell takes binary states. The updated state is determined by 3 cells, i.e., two neighboring cells and its own, and hence there are 2^{2^3}=256 possible evolution rules which can be labeled from Rule 0 to Rule 255. The time-evolution of cell states can be written by

$$x_i(k) = F[x_{i-1}(k-1), x_i(k-1), x_{i+1}(k-1)]. \quad (1)$$

Here, $\boldsymbol{x}(k)$=$\{x_1(k), x_2(k), \cdots, x_N(k)\}$ is an N-dimensional node state vector and $x_i(k)$ is the state of i-th node at time step k, where $i \in \{1, 2, \cdots, N\}$ and $k \in \{1, 2, \cdots, M\}$.

C. Cellular Automata Applied on Reservoir Computing

To apply CA on RC, the inputs should be converted into binary format with thermometer encoding, which has been proved to increase the robustness of the neural network significantly [11]. Decomposing the original images u in d binary channels, we can obtain the l-th channel of the input $u^{(l)}$, where $l \in [1, d]$. There is no communication between binary channels.

After obtaining the binarized input, the ECA rule is applied to rows and columns independently with a fixed boundary condition. These two image results are combined with a bitwise XOR operation [4]. This process is repeated for M times. Let $g^k(x)$ be the state after applying ECA evolution rules on x for k times. And $x^{(l)}(k)$ is a boolean time-dependent image, which is given by

$$x^{(l)}(k) = g^k(u^{(l)}) = \begin{cases} u^{(l)}, & k = 0, \\ g^1(x^{(l)}(k-1)), & k > 0. \end{cases} \quad (2)$$

We can obtain the state of the reservoir in the i-th position, k-th iteration and l-th binary channel as $x_i^{(l)}(k)$. Thus,

$$x^{(l)}(k) = [x_1^{(l)}(k), x_2^{(l)}(k), ..., x_N^{(l)}(k)]. \quad (3)$$

The images are iterated by rows and columns independently. Defining $x_{row}^{(l)}(k)$ as the results of iterating images by rows, i.e., the state of each updated cell is determined by two horizontal neighboring cells and itself. Similarly, $x_{col}^{(l)}(k)$ represents the results of iterating images by columns. Vectors $x_{feature}^{(l)}(k)$ are obtained by combining $x_{row}(k)$ and $x_{col}(k)$ with an XOR operation

$$x_{feature}^{(l)}(k) = x_{row}^{(l)}(k) \oplus x_{col}^{(l)}(k). \quad (4)$$

After this, we apply a max-pooling layer to improve the generalization of the network and reduce the weights in the classifier. By binarizing internal states, CA is suitable for hardware implementation. However, the classifier still requires softmax operation in [4], which should be eliminated for the ease of implementation on resource-constrained devices.

D. Bloom Filter

To completely eliminate the FP calculations, we replace the softmax-function with the Bloom filter which is proposed in [6]. Bloom filter is considered as a space-efficient probabilistic data structure based on random access memory (RAM), which aims at testing whether an unknown element is a member of the given set. As the term "probabilistic" suggests, the query result may contain errors, i.e., Bloom filter returns either "possibly in the set" or "definitely not in the set." The most significant advantage of the Bloom filter is its memory space efficiency over other data structures, which is suitable for error-resilient applications like machine learning [12].

In the standard Bloom filter model, the element is considered as an L bits binarized vector, which represents the address within the Bloom filter. Defining $B(index)$ as the $index$-th bit in the Bloom filter, where $index \in [0, 2^L - 1]$.

At the insertion phase, all the values in Bloom filters are set to zero initially. Each training sample is inserted into the corresponding Bloom filter based on their labels. The value of accessed bit $B(index)$ is set to one. At the query phase, the testing sample is sent to all the Bloom filters and returns the value of accessed bit $B(index)$ in every Bloom filter. When it returns positive, the testing sample is considered as "possibly in this category". When it returns negative, this sample is judged as "definitely not in this category."

III. Proposed Method

A. BloomCA

As a novel RC architecture, *BloomCA* is an ingenious combination of CA and ensemble Bloom filter. Fig. 1 shows the overview of *BloomCA* which consists of CA for extracting a high-dimensional binary feature vector from an input image and each Bloom filter corresponds to a class label. Once an input image is provided, *BloomCA* firstly extracts the binarized feature vector in the same way as Sec. II-B. Then, the similarity between the extracted feature vector and each Bloom filter is computed and the class whose corresponding Bloom filter exhibits the maximum similarity is outputted as the model prediction. In the following, we detail the algorithm of *BloomCA* which exploits the benefit of CA and Bloom filter.

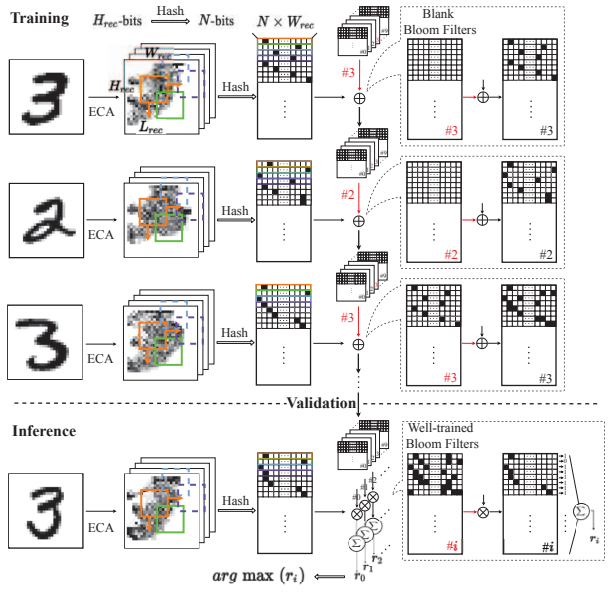

Fig. 1: Overview of *BloomCA*.

Training Phase: Initially, the Bloom filter classifier is blank, i.e., all the values are set to zero. Then, the input image is fed into the CA part for feature extraction. In the next step, we extract patches of the CA output by applying a $W_{rec}\times H_{rec}$ size receptive field with a stride of L_{rec} and apply a simple hash function to each extracted patch to obtain a binary feature vector. The detail of the hash function is provided in Sec. III-B. Finally, the extracted feature vector is inserted into the corresponding Bloom filter, which is specified by the training label, e.g., a training image labeled "5" is supposed to be inserted into the #5 Bloom filter.

Inference Phase: Similar to the training phase, the input images are fed into the CA followed by image patch extraction and application of the hash function to obtain the feature vector for the input image. Then, using counters and bit-wise AND gates to calculate the similarity between the feature vector and Bloom filters, the Bloom filter with the highest response is chosen as a representative category of the testing image.

Although the pattern dynamics are extremely elevated by the CA reservoir, it also increases the memory usage of Bloom filters in every category. The more elements are added to one single Bloom filter, the higher probability of false positives (FPR) [12]. To optimize this data structure, we propose utilizing the ensemble Bloom filter as the classifier in our RC system.

B. Ensemble Bloom Filter

Ensemble learning is a machine learning paradigm where multiple base learners are trained to solve the same problem. The generalization ability of an ensemble is usually much stronger than the base learners. Base learners are usually generated by a base learning algorithm which can be the decision tree, neural network, or other machine learning algorithms [13]. The predictions from the base learners are combined with "meta-algorithm" to yield the final prediction. In our case, we apply the bagging algorithm. The training dataset is divided into N_{sub} subsets and inserted into different Bloom filters as base learners. Then the results of these base learners are summed up together, which can be considered as an ensemble Bloom filter classifier.

For ensemble Bloom filter, a binarized pattern having $N \cdot L$ bits is split into N vectors of L bits. Defining $B_n(i)$ as the i-th bit in the n-th mini Bloom filter, where $n \in \{1, 2, ..., N\}$. In our case, the patterns from the $W_{rec}\times H_{rec}$ size receptive field need to be memorized by a different ensemble Bloom filter. Similar to [5], each binarized pattern is split into W_{rec} and H_{rec} vectors in row and column. During the insertion phase, the vectors are inserted into the corresponding Bloom filters, which we call "mini Bloom filter". These vectors are considered as the addresses within the mini Bloom filters as well as the results of hash function. When it comes to the query phase, only if all the values of accessed bit $B_n(i)$ in mini Bloom filters are positive (logic "1"), this Bloom filter returns one. Otherwise, this Bloom filter returns zero. All the returned results of Bloom filters in each category are summed up together and considered as discriminator response r, which also represents the similarity between the testing sample and the corresponding category. The discriminator with the highest response r is chosen as the representative category.

After training each Bloom filter, we select those exhibiting good classification accuracy by using validation samples. Similar to the training phase, the class labels of the validation images are predicted to evaluate the classification accuracy of each Bloom filter. Then, we select N_{inf} Bloom filters exhibiting the top N_{inf} performance, which construct the classifier used in the inference phase.

Instead of using a single standard Bloom filter with high memory cost, utilizing Bloom filters in an ensemble way can effectively improve the performance in image recognition tasks. Meanwhile, the number of well-trained Bloom filters in each Bloom filter pool is decreased, which leads to a significant reduction in memory cost during the inference phase.

IV. Experiment

A. Experiment Setup

In our experiment, we focus on the MNIST dataset which is a collection of 70k handwritten digits in grayscale format [14]. Among 60k images, we randomly select 55k images as the training set and 5k images as the validation set. Dividing the training set into N_{sub} subsets for every category in order. The images in the training set are used to populate Bloom filters while the validation images are used to optimize the hyperparameters such as the CA evolution rules. The rest of the 10k images are used for testing.

In our experiment, w=28, h=28, d=16, R=256, N_{inf}=96. The iteration times M is in the range from 1 to 24. The max-pooling layer is selected to have a stride of 2, a squared window of size 2, and zero paddings. A 5×5 receptive field is applied to every binary channel and iteration of the reservoir output, with a stride of 3. Similar to [15], we only choose the first iterative pattern which represents the original input and the

TABLE I: Performance Comparison

	Bloom WiSARD (baseline)	ensemble Bloom filter	*BloomCA* (proposed)
Arithmetic	multiplication	addition	addition
Number of hash	3	1	1
Accuracy(%)	91.50	89.58	91.86
Inference Memory Cost(KB)	819.05	18.75	18.75

latest iterative pattern as features. This strategy can effectively refine the input feature for the ensemble Bloom filter and reduce memory cost in both the training and inference phase.

B. Experiment Result

Using the *BloomCA* model described in section III-A, we check the performance of different CA evolution rules. According to the hyperparameter optimization from the validation set, we adopt the evolution Rule 184 and iteration times $M{=}8$.

According to the evaluation results, the accuracy is increasing with the training memory cost scaling. When the training memory cost is lower than 300KB, the accuracy begins to drop down sharply. In this case, too many features are inserted into the same Bloom filter, which makes the Bloom filter polluted. When the training memory cost is up to 825KB, the trend of accuracy appears to be saturated.

Table I shows the results of several different models. Comparing with the baseline *Bloom WiSARD*, the *BloomCA* achieved $819.05/18.75{\approx}43\times$ memory reduction for the inference phase, which mainly comes from using Bloom filters in an ensemble way. Although this method also leads to a slight decrease in accuracy, such a disadvantage can be overcome by using CA as the reservoir. The rich pattern dynamics recreated by CA can effectively provide more candidates for selection and promote the accuracy from 89.58% to 91.86%, which also illustrates the impact of CA. Overall, in comparison with another state-of-the-art model *Bloom WiSARD* [5], *BloomCA* significantly reduces the memory cost for the inference phase without hurting the accuracy.

C. Hardware Implementation

The hardware architecture is synthesized in 65-nm ASIC flow. Table II shows the comparison between [5] and *BloomCA* in terms of ASIC area and energy consumption. The maximum propagation delay is 3.78 nanosecond. The memory takes $81.73/168.64{\approx}48.46\%$ of the area in *BloomCA* while the other circuits take 51.54%. When it comes to power consumption, the memory part only takes $5.15/126.9{\approx}4.1\%$ and the percentage of other circuits is increased to 95.9%.

Although the memory occupies almost half of the whole circuit area, the energy is mostly consumed by the non-memory part, i.e. CA, due to the high switching activity of the non-memory part. Note that we may underestimate the hardware resource cost of [5], since we only evaluate the memory part of [5] for ease of comparison. Hence, the performance gain of *BloomCA* will further increase if we take into account the control circuits and ALU of [5].

TABLE II: Hardware Performance Comparison

	Bloom WiSARD	*BloomCA*	
	Memory	Memory	Total
Area (kGates) reduction	3570.35	81.73 (1/43)	168.64 (1/21)
Power (mW) reduction	225.14	5.15 (-97.71%)	126.9 (-43.64%)

V. Conclusion

In this work, we proposed a novel RC architecture *BloomCA*, which is a combination of the reservoir using CA and ensemble bloom filter classifier. Achieving a 43x reduction in the inference memory cost while maintaining accuracy. Our hardware implementation also demonstrates that *BloomCA* achieves over $21\times$ and 43.64% reduction in area and power, respectively.

The experiment result also illustrates the impact of using CA as a reservoir. Mapping the input signal into high dimensional feature space, the rich dynamics recreated by CA can effectively improve the performance of the ensemble bloom filter classifier. This model can completely eliminate FP calculation and integer multiplication. Due to its simplicity, *BloomCA* shows promising potential for hardware implementation.

Acknowledgment

This work was partially supported by JST, PRESTO Grant No. JP-MJPR18M1, JSPS KAKENHI Grant No. 18K13800, and the NEC Corporation.

References

[1] B. Schrauwen, D. Verstraeten, and J. Campenhout, "An overview of reservoir computing: theory, applications and implementations," in *ESANN*, 2007.

[2] E. A. Antonelo and B. Schrauwen, "On learning navigation behaviors for small mobile robots with reservoir computing architectures," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, pp. 763–780, 2015.

[3] Ö. Yılmaz, "Machine learning using cellular automata based feature expansion and reservoir computing," *J. Cell. Autom.*, vol. 10, pp. 435–472, 2015.

[4] A. Morán, C. F. Frasser, and J. L. Rosselló, "Reservoir computing hardware with cellular automata," *ArXiv*, vol. abs/1806.04932, 2018.

[5] L. Santiago, L. Verona, F. Rangel, F. Firmino, D. Menasché, W. Caarls, M. Breternitz, S. Kundu, P. Lima, and F. M. G. França, "Weightless neural networks as memory segmented bloom filters," *Neurocomputing*, 2020.

[6] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, pp. 422–426, 1970.

[7] H. Jaeger, "Adaptive nonlinear system identification with echo state networks," in *NIPS*, 2002.

[8] T. Natschläger, W. Maass, and H. Markram, "The "liquid computer": A novel strategy for real-time computing on time series," 2002.

[9] Y. Kume, S. Bian, and T. Sato, "A tuning-free hardware reservoir based on mosfet crossbar array for practical echo state network implementation," *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 458–463, 2020.

[10] M. Lukosevicius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Comput. Sci. Rev.*, vol. 3, pp. 127–149, 2009.

[11] J. Buckman, A. Roy, C. Raffel, and I. J. Goodfellow, "Thermometer encoding: One hot way to resist adversarial examples," in *ICLR*, 2018.

[12] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *IEEE Communications Surveys & Tutorials*, vol. 14, pp. 131–155, 2012.

[13] Z.-H. Zhou, *Ensemble Learning*. Boston, MA: Springer US, 2009, pp. 270–273.

[14] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[15] A. Lopez, J. Yu, and M. Hashimoto, "Low-cost reservoir computing using cellular automata and random forests," *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, 0 (to appear).