Resistive Bloom Filters: From Approximate Membership to Approximate Computing with Bounded Errors

Vahideh Akhlaghi*, Abbas Rahimi†, Rajesh K. Gupta*
*Department of Computer Science and Engineering, UC San Diego, La Jolla, CA, USA
†Department of Electrical Engineering and Computer Sciences, UC Berkeley, Berkeley, CA, USA
{vakhlagh, gupta}@cs.ucsd.edu, abbas@eecs.berkeley.edu

***Abstract*—Approximate computing provides an opportunity for exploiting application characteristics to trade the accuracy for gains in energy efficiency. However, such opportunity must be able to bound the error that the system designer provides to the application developer. Space-efficient probabilistic data structure such as Bloom filter can provide one such means. Bloom filter supports approximate set membership queries with a *tunable rate* of false positives (i.e., errors) and no false negatives. We propose a resistive Bloom filter (ReBF) to approximate a function by tightly integrating it to a functional unit (FU) implementing the function. ReBF approximately mimics partial functionality of the FU by recalling its frequent input patterns for computational reuse. The accuracy of the target FU is guaranteed by bounding the ReBF error behavior at the design time. We further lower energy consumption of a FU by designing its ReBF using low-power memristor arrays. The experimental results show that function approximation using ReBF for five image processing kernels running on the AMD Southern Islands GPU yields on average 24.1% energy saving in 45 nm technology compared to the exact computation.**

I. Introduction

The scaling of physical dimensions in semiconductor circuits opens the way to an astonishing over 7 billion transistors on massively parallel integrated architectures enforcing energy efficiency as a primary concern [1]. Emerging applications including graphics, multimedia, recognition, and data mining offer significant degrees of tolerance to the precision in computing results, thus enabling the emerging vision of "approximate computing". Approximate computing provides higher energy efficiency by accepting errors for a specific application. As an example, approximate adders are introduced by reducing circuit complexity [2], [3]. These techniques typically consider a specific set of input patterns to design approximate hardware/circuit, hence they cannot guarantee error bounds for all data set.

Another emerging, and unrelated, development is growing use and capability of non-volatile memory (NVM) in systems. NVM today offers high density and near zero leakage power that enables energy-efficient computation on *memory-centric* architectures. Resistive RAM (ReRAM), in particular, offers low-write energy, fast read access time and low read voltage [4]. A 1-Mb ternary content addressable memory (TCAM) test chip is proposed as an energy-efficient memory-centric architecture providing $> 10\times$ smaller cell size compared to SRAM-based TCAMs [5]. Such properties determine memristors are a promising replacement for traditional SRAM-based memories in low-power and high capacity designs.

Value locality and similarity inside various applications [6] especially in data-level parallel applications provide opportunity to reuse frequent computations [7], [8], thereby, improving energy-efficiency by avoiding redunadant processing. In computational reuse, memories play an important role in maintaining pre-calculated computations. In this context, a Bloom Filter (BF) provides a data structure with the ability to provide probabilistic response to membership queries. A BF stores a set of elements (i.e., patterns) using multiple hash functions performed on them. The size of the memory required for a BF scales linearly with the number of stored elements in the filter [9]. BF data structure is extensively used to keep track of incoming data and states of flows in network applications [10], [11].

Thus, our working thesis is that a combination of approximation in computing with enhanced reuse methods enabled by NVM can yield high energy efficiency. In this paper, we explore this possibility for programmable parallel architectures. In particular, this work aims to increase computational reuse for approximate computing with bounded errors by designing BFs using non-volatile memory elements. We make the following contributions: I) We design BFs to generate an approximate function with a guaranteed error bound. Hence, a set of BFs is tightly-coupled to individual FU to approximately represent highly frequent computations of the associated FU. Each BF reports no false negatives (i.e., recall rate of 100%), and has a *tunable* parameter to control false positives. II) To further lower energy consumption and enable scalability, we utilize low-power memristor array in designing resistive Bloom filter (ReBF). III) We integrate ReBF into the Southern Islands GPU that yields on average 24.1% energy saving for five image processing kernels. Caltech 101 computer vision data set [12] is used for profiling and finding the error bounds.

The rest of the paper is organized as follows. Section II explains our proposed methodology to use BF in approximating a function with the bounded error rate. The proposed scalable BF to demonstrate additional functionality is presented in Section III. The experimental results are discussed in Section IV. Finally, Section V concludes the paper.

II. Approximate Computing with Bloom Filters

A. Bloom Filters (BFs)

BFs provide a compact representation of a set of elements. A BF consists of an m-bit vector which is programmed using k random hash functions. For a given element, k bits of the Bloom vector (BV), selected by the k hash functions, are set in the programming stage. For a look up process, the same hash functions are computed on an input pattern. If the k bits in the vector determined by k hash values are all set, the input pattern is said to be present in the BF. Since those k values may also be obtained by any of actual members, the presence of an input in the BF may be confirmed erroneously. On the other hand, if at least one of the k bits is not set, the input is certainly not the member of the BF. This explains that a BF has a recall rate of 100% and there is no false negative; however, it allows tunable false positives (FP), the rate of which is formulated as:

$$fp = (1 - e^{-\frac{nk}{m}})^k \tag{1}$$

where, n is the number of patterns saved in the BF, m is the length of BV, and k is the number of hash functions.

The hardware implementation of the BF is represented in Figure 1. The BV is shown as an m-bit array. For a look up operation, BF, at first, computes k hash functions (HF) concurrently. The input data is also split into the number of bytes to parallelize the computations of each hash function. Each HFB module inside the HF performs computation on one byte of the data. Each HFB is made of XOR gates, and a number of coefficients, each has $log_2(m)$ bits. Coefficients are XORed if the corresponding bit of the input is one. The last XOR gate in each HF produces the final address. If all of the k bits of the BV are one, hit occurs and the EnL signal becomes one.

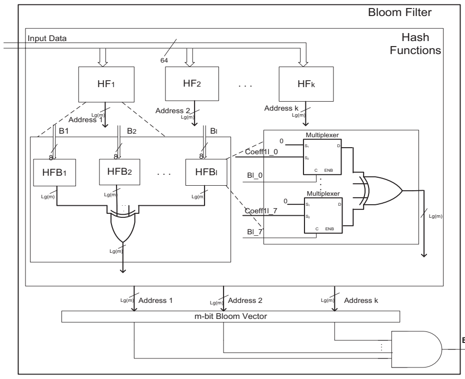

Fig. 1: The hardware architecture of Bloom filter.

B. Utilizing Bloom Filters for Function Approximation

In this section, we describe our proposed method that exploits BF to expand the probabilistic membership query to function approximation.

To avoid redundant computation overhead due to re-execution of an operation for the same inputs, we identified highly frequent output values in the function co-domain and stored their corresponding input patterns in a set of BFs. To perform a function for a given input, at first, the membership of the input is queried in the BFs associated to the function. If a hit event occurs, the corresponding output value is returned as the final output of the function, and all pipeline stages in the implementation of the functional unit (FU) are clock-gated to save energy. As described in Section II-A, positive responses of the BF to the membership queries are not always correct due to the FP. The source of error in our computation, is a FP event where the BF wrongly reports the *output value* as the function output. Here, the rate of the error corresponds to the FP rate, and can be controlled by tuning the parameters affecting the FP (i.e., *n, m, k*). In case of a miss, the FU continues the exact computation for the mismatched input patterns. Based on the aforementioned details, a set of BFs can be exploited to approximate partial outputs of a function with the bounded errors that provide guaranteed quality. The output quality, here, is maintained by adjusting the FP rate of BFs.

C. Bounding Bloom Filter Errors at Design Time

In our method, we are able to limit the error rate (i.e., the degree of approximation) to meet the acceptable output quality by limiting the FP rate of associated BFs at the design time.

TABLE I: Maximum Acceptable Error Rate and Output PSNR

App	ADD	MUL	MAC	SQRT	PSNR (min, max, avg) (dB)
Sobel Filter	0.001	0.0001	0.001	0.001	(26.4,32.9,28.0)
Sharpen	–	0.01	0.01	0.01	(24.7, 36.8, 28.12)
Roberts	–	0.001	0.001	0.001	(25.4, 32.5, 27.9)
Prewitt	–	0.01	0.001	0.01	(25.2, 33.3, 27.1)
Scharr	–	0.001	0.0001	0.001	(26.1, 31.1, 27.2)

We focus on multimedia applications that are amenable to approximation. The multimedia applications offer the well-known notion of output quality with peak signal-to-noise ratio (PSNR). With approximate computing, an application with PSNR of equal or greater than 25 dB can still appear to execute correctly from the user's perspective [13]. FUs in the GPU architecture are targeted for integrating BFs. To ensure that the output quality will never go below the desired threshold, we need to configure BFs. We require, at first, to identify the maximum tolerable error rate for each FU in a given kernel. We set the error magnitude conservatively to its maximum value for each FP event during the design time analysis. Then, the parameters of BF should be set to yield the FP rate less than or equal to the error rate. To do that, we need to specify the number of inputs to store in the BF through profiling. For the sake of clarity, we describe the process of configuring BFs in more detail, given the fact that four types of operations are identified in GPU architecture: adder (ADD), multiplier (MUL), multiply-accumulator (MAC) and SQRT.

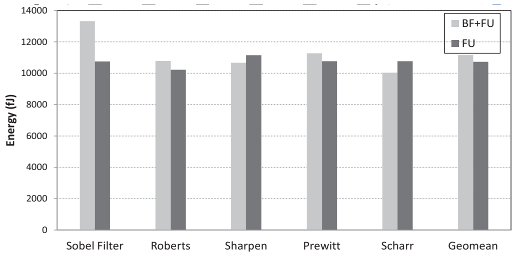

Fig. 2: Energy consumption comparison of the proposed architecture using CMOS BFs and conventional FU.

Maximum Tolerable Error Rates. To find maximum tolerable error rates of the FUs used in a kernel, we use the following algorithm, and simulate the kernel for 30 different images using Multi2Sim [14]. The desired output quality, here, is assumed to be equal or greater than 25 dB. However, the algorithm has the ability to find the maximum error rate for any given PSNR threshold.

The first step is to find the maximum tolerable error rate for each individual operation. To achieve this, each time, one operation is selected and random error with different rates is injected into it. We start from error rate of 0.1 and decrease it until the average PSNR of final output images becomes acceptable. The next step aims to inject errors to all FUs to see their combined effect on the output quality. Here, we inject errors simultaneously into all operations with the corresponding error rate obtained in the previous step. In case of unacceptable output quality, in the third step, we identify the top frequent operation using the assembly code of the kernel generated by Multi2Sim, and decrease its error rate by ten times. The process is repeated for the next frequent FU until the acceptable PSNR is achieved. Table I summarizes the maximum tolerable error rate obtained for each FU of five image processing applications, and the corresponding PSNR.

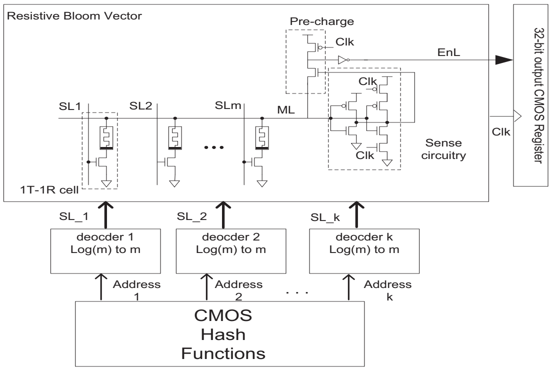

Fig. 3: 1T-1R implementation of ReBF.

The Number of Input Patterns Saved in BF. To specify the frequent outputs and their corresponding input patterns, we performed profiling for every FU in the kernel using 10% of the training samples. If we want to save all inputs of each frequent output value, large size of BF is needed. To overcome this problem, we save a few number of input patterns for an output which constitutes most fraction of the hit rate achieved by that output (e.g., one (329) input(s) for the most frequent output value of MAC in Sobel leads to hit rate 6.2% (7.3%)).

BF Configuration. After specifying the FP rate of BFs and the number of elements to save (i.e., *n*) in the profiling stage, we change the value of *m*, and *k* to meet the pre-determined error rates and design a BF for each operation to investigate its energy overhead.

III. ReBF: Resistive Bloom Filter

Before introducing architecture of a ReBF, let us examine energy efficacy of CMOS implementation of BFs. We implement BFs configured for five image processing applications using Verilog, and synthesized them with 45 nm standard CMOS library. Each individual output value is stored in a 32-bit register, connected to the corresponding BF. Figure 2 illustrates the energy of the proposed architecture using CMOS BFs compared to the conventional one. As shown, computational reuse with the aid of CMOS BFs offers negligible or no benefit (i.e., on overage the overhead is 4.4%). This energy overhead is mainly due to the implementations of m-bit BVs (i.e., about 70% of the power). To improve the energy efficiency of our proposed architecture, we use 1-transistor/1-resistive (1T-1R) cells, to implement the m-bit BV.

A. ReBF Architecture

The architecture of ReBF is shown in Figure 3. Each bit of the resistive Bloom vector is implemented by a 1T-1R cell. The cells are connected to each other through a match line (ML). The stored value in each cell is represented by the value of the resistor element. High value of the resistor is considered as digital one, and low value as digital zero. The transistor in the cell is controlled by a select line (SL). During the programming stage, *k* hash values for each member are obtained using the combinational CMOS circuit, and, then, the resistor of the corresponding cell in ReBF is set to the high value.

TABLE II: Energy Consumption of Bloom vector

Size (bits)	**2048**	**1024**	**512**	**256**	**128**	**64**
Vdd (V)	1.0	0.71	0.6	0.54	0.51	0.47
RBV (fJ)	19.3	17.12	8.04	5.10	3.77	1.62
CMOS BV (fJ)	12188.4	6662.4	3992.4	2603.1	1837.35	1649.55

For the look up/search process, the ML is, at first, pre-charged to Vdd via the pre-charge circuit. Then, the same *k* hash functions generate *k* addresses which will be decoded in parallel through *k* CMOS decoders. The select lines of the corresponding *k* cells turn on the NMOS transistor in the cell. If the stored value in any of the *k* cells is zero (i.e., the resistor value is low), a path between Vdd and ground will be established and the ML will be discharged. The voltage drop on the ML will be amplified by the sense circuit; therefore, a full swing will be observed on the EnL signal. This means that the incoming input is not saved in the ReBF. In this case, EnL prevents the output register to be read. The delay of the circuit is determined by the time it takes to pre-charge ML to Vdd, and the time it takes to discharge the line and observe the full swing on EnL. The worst-case delay happens when only one of the cells is not matched, and tries to discharge the ML. If more than one cell are not matched, the ML will be discharged rapidly, thereby, the delay is lower than the previous case. However, if all *k* cells are set to one, the resistor values are all high, and prevent the ML from discharging. This means that hit occurs and EnL signal becomes high, which leads to returning pre-calculated value stored in a CMOS register as the final output of the FU that uses the ReBF.

IV. Experimental Results

A. Experimental Setup

To assess the efficiency of the function approximation using ReBF, we choose five image processing applications adopted from AMD APP SDK v2.5 [15]: Sobel, Robert, Prewitt, Scharr and Sharpen. The openCL code of these applications are simulated by Multi2Sim [14] to perform profiling and finding the error rate bound on four FUs. We generates the VHDL code of these FUs as the six-stage pipeline unit, commensurate with the AMD Southern Islands GPUs [14], using FloPoCo [16]. The hardware realization of different size of BFs with different number of hash functions are implemented using Verilog. To calculate the energy consumption, the implemented FUs and BFs are synthesized using Synopsys Design Compiler, with 45 nm standard CMOS library. The operating voltage is set to 1.0V and the clock period is 1.5 ns. To estimate the power and delay of ReBF, transistor-level design of the Bloom vector of different sizes is performed using HSPICE. We consider R_{ON} as 10K Ω and R_{OFF} as 1M Ω and set the other parameters of the cell based on the one presented in [17].

B. Energy Saving

Energy consumption of an individual resistive Bloom vector (RBV) and CMOS Bloom vector of different length of

TABLE III: Optimum ReBF configuration for different applications

FU	Sobel					Roberts					Sharpen					Prewitt					Scharr				
	#Out	#In	HR%	BV	#Fn	#Out	#In	HR%	BV	#Fn	#Out	#In	HR%	BV	#Fn	#Out	#In	HR%	BV	#Fn	#Out	#In	HR%	BV	#Fn
ADD	2	12	29.46	256	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
MUL	2	16	29.61	1024	3	6	26	41.61	512	4	6	15	42.18	256	2	8	30	59.4	512	2	4	12	27.7	256	4
MAC	4	12	25.44	256	4	10	12	29.5	256	4	6	15	30.2	256	2	6	15	32.6	1024	2	8	18	37	512	4
SQRT	16	16	20.8	512	3	10	10	29.5	256	4	-	-	-	-	-	14	14	82	256	2	18	18	10	512	3

2048, 1024, 512, 256, 128, and 64 bits are summarized in Table II. The delay of the RBV of different size is fixed at 1.4 ns by adaptively adjusting the operating voltage. As we can see, implementing Bloom vector using memristor cells presents extremely low energy consumption even if the larger vector is used. This presents the scalability of RBV and its benefit in representing more functionality. To assess the efficiency of our approach, we profile five image processing kernels using 10% of Caltech 101 computer vision dataset [12], and find the maximum tolerable error rate for each FUs in the kernels as described in Section II-C using 30% of the dataset. For each kernels, we select various hit rates obtained from profiling stage. Then, we configure BFs based on the number of elements that depends on the hit rate (HR), and the maximum error rate obtained for each FU. We evaluate energy consumption of our approach for each configuration to find the maximum energy improvement compared to the conventional FU without ReBF. For each application, we summarize the optimum configuration of the Bloom filters (e.g., the length of Bloom vector (BV) , and the number of hash functions (Fn)) integrated to each FU in Table III.

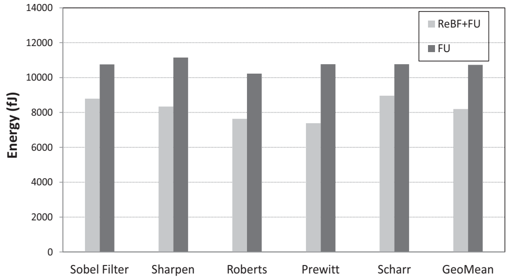

Fig. 4: Energy comparison of the proposed architecture using ReBFs and conventional FUs.

Figure 4 illustrates the total energy of using ReBF (including resistive and CMOS parts) along with FUs and energy of conventional FUs without ReBF. ReBf demonstrates 21.7%, 25.2%, 25.3%, 31.4%, and 16.8% energy improvement for Sobel filter, Sharpen, Roberts, Prewitt, and Scharr, respectively. As an example, for Sharpen, maximum improvement is achieved by using ReBFs for MUL and MAC, to store 15 patterns for each module. To meet the maximum error rates, we employ Bloom vector of size 256 bits, and two hash functions for each operation. The provided hit rates are 42.18% and 30.2% for MUL and MAC, respectively. Here, high energy consumption of CMOS modules inhibits us from further computational reuse and energy saving.

V. Conclusion

Modern multimedia applications offer massive parallelism and significant degrees of tolerance to approximate computing. This paper aims to address the following challenge: *how to increase approximate computational reuse through non-volatile resistive storages in GPUs with bounded errors?* Combining computational reuse and approximate computing, we propose a resistive Bloom filter (ReBF) that provides an approximate representation of a function by integrating Bloom filters to the hardware functional units. BeBFs are used to store highly frequent patterns to avoid re-executions. This methodology has the ability to control the degree of function approximation by adjusting the false positive rate of the ReBFs. To reduce energy consumption of the proposed architecture, low-power memristive arrays are exploited to perform search operations at extremely low energy. Experimental results show that our approach represents on average 38.42% of the functionality of FUs in five different kernels running on GPUs, while guaranteeing the acceptable outputs with PSNR of greater than 27 dB. This leads to on average 24.1% energy reduction compared to conventional architectures without ReBF.

VI. Acknowledgements

This work was supported by NSF's Variability Expedition (1029783).

References

[1] Maxwell geforce gtx titan x. [Online]. Available: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-x

[2] V. Gupta, et al., "Low-power digital signal processing using approximate adders," *IEEE TCAD*, vol. 32, no. 1, pp. 124–137, 2013.

[3] V. Gupta, et al., "Impact: imprecise adders for low-power approximate computing," in *IEEE/ACM ISLPED*, 2011, p. 409414.

[4] M.-F. Chang, et al., "19.4 embedded 1mb reram in 28nm cmos with 0.27-to-1v read using swing-sample-and-couple sense amplifier and self-boost-write-termination scheme," in *IEEE ISSCC*, Feb 2014, pp. 332–333.

[5] J. Li, et al., "1 mb 0.41 μm^2 2t-2r cell nonvolatile tcam with two-bit encoding and clocked self-referenced sensing," *IEEE JSSCC*, vol. 49, no. 4, pp. 896–907, 2014.

[6] M. H. Lipasti, et al., "Value locality and load value prediction," in *IEEE ASPLOS*, New York, NY, USA: ACM, 1996, pp. 138–147.

[7] A. Rahimi, et al., "Spatial memoization: Concurrent instruction reuse to correct timing errors in simd architectures," *IEEE TCAS*, vol. 60, no. 12, pp. 847–851, Dec 2013.

[8] A. Rahimi, et al., "Approximate Associative Memristive Memory for Energy-efficient GPUs," *IEEE DATE*, 2015, pp. 1497–1502.

[9] S. Dharmapurikar, et al., "Deep packet inspection using parallel bloom filters," in *IEEE HotI*, 2003, pp. 44–51.

[10] H. Song, et al., "Fast hash table lookup using extended bloom filter: an aid to network processing," in *SIGCOMM*, 2005, pp. 181–192.

[11] F. Bonomi, et al., "Beyond bloom filters: from approximate membership checks to approximate state machines," in *IEEE SIGCOMM*, 2006, pp. 315–326.

[12] Caltech 101. [Online]. Available: http://www.vision.caltech.edu/Image_Datasets/Caltech101/

[13] M. Barni, *Document and Image Compression*. CRC Press, 2006.

[14] Multi2sim: A heterogeneous system simulator. [Online]. Available: https://www.multi2sim.org/

[15] Amd app sdk v2.5. [Online]. Available: http://www.amd.com/stream

[16] Flopoco: Floating-point cores generator. [Online]. Available: http://flopoco.gforge.inria.fr/

[17] M. Zangeneh, et al., "Design and optimization of nonvolatile multibit 1t1r resistive ram," *IEEE TVLSI*, vol. 22, no. 8, pp. 1815–1828, Sept 2013.