Accelerating Stochastic Computation for Binary Classification Applications

Lezhong Huang†, Guanhui Chen†, Peng Li§, Weikang Qian†*
†University of Michigan-Shanghai Jiao Tong University Joint Institute,
Shanghai Jiao Tong University, Shanghai, China.
§Storage Technologies Group, Intel Corporation, Hillsboro, OR, U.S.A.
*Email: qianwk@sjtu.edu.cn

***Abstract*—Stochastic computation is a non-conventional computation paradigm, which uses digital circuits to operate on stochastic bit streams. Although it has advantages such as strong fault tolerance and low hardware cost, its drawback is its long computation time. In this work, we target at stochastic computation used in binary classification applications, such as image segmentation and pattern classification, and propose a novel accelerating module. We study how the design parameters affect the error rate and computation time. We further propose how to find the optimal design parameters. A case study on an image segmentation algorithm shows the effectiveness of our proposed solution.**

***Index Terms*—stochastic computation, acceleration, error rate**

I. Introduction

As CMOS devices scale into the nanometer regime, they become more and more susceptible to process, voltage, and thermal variations [3]. Furthermore, soft error can result in transient malfunction of circuits [14]. Circuit designers are now faced with the challenge of how to design reliable circuits from unreliable devices.

Stochastic computation, a method to perform computation using digital circuits that operate on stochastic bit streams, is highly tolerant of bit flip errors and hence, offers a solution to the reliability problem [9]. In stochastic computation, conventional digital circuits are still used to process digital signals. However, the way to encode a value through zeros and ones is different from conventional binary radix encoding. Here, a real value x in the unit interval is represented by a stream of *random* bits $X_1, X_2, \ldots, X_l \in \{0, 1\}$, with each X_i having probability x of being a one and probability $(1-x)$ of being a zero, i.e., $P(X_i = 1) = x$ and $P(X_i = 0) = 1 - x$. For example, both the streams A and B in Fig. 1 represent the value $4/8$.

Stochastic computation has strong tolerance to bit-flip errors, because a bit flip occurring anywhere in a stream only slightly changes the value encoded by the stream. In contrast, for the binary radix encoding, bit flips occurring at the most significant bits can cause a large error in the value. With stochastic encoding, many complex arithmetic operations, such as multiplication, division, and square root operation, can be implemented with very simple digital circuits [4], [7], [12]. For example, Fig. 1 shows that a single AND gate can perform multiplication: the probability of obtaining a one in the output stream equals the product of probabilities of obtaining ones in the two input streams. In contrast, conventional multiplier operating on binary radix needs a very complex logic design [10].

Due to its low hardware cost, stochastic computation has been used in some hardware-demanding applications, such as real-time image processing [1], low-density parity-check decoding [11], and artificial neural networks [5].

The major drawback of stochastic computation lies in its inefficiency in encoding: to encode a value with a precision of $\frac{1}{2^n}$, a bit stream of length 2^n is required. Traditionally, the value encoded by a stochastic bit stream is obtained by counting the total number of ones in the entire bit stream; this leads to a long computation time. However, in some binary classification applications, such as image segmentation [8] and pattern classification [2], what we are interested in is whether the final result is larger than a threshold. If we apply stochastic computation to realize these applications, then we do not need to find the exact probability of a one in a stochastic bit stream; we only need to know whether that probability is larger than a threshold x or not. In this case, we actually do not need to examine the entire sequence to reach a conclusion; sometimes, examining a partial segment is enough. For example, suppose that we want to determine whether the value encoded by a stochastic bit stream is larger than the threshold 0.5. If we find that the first 10 bits of the stream contain 9 ones, then it is highly likely that the probability of a one in the entire stream is larger than 0.5. Then, we can reach a conclusion in a much shorter time than the traditional approach of counting the entire sequence.

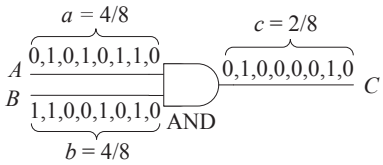

Fig. 1: An AND gate performs multiplication on values encoded by stochastic bit streams.

In this work, we apply the above idea and propose a novel module to accelerate the stochastic computation used in binary classification applications. It divides a stochastic bit stream into a number of segments of the same length and checks each segment one by one to infer whether the probability of a one in the entire stream is above or below 0.5. Although the basic idea is simple, the challenging part is how to choose the design parameters including the segment length and the threshold value. These two parameters will affect the error rate and the computation time of the whole system. We give a rigorous mathematical analysis of how the design parameters affect the error rate and the computation time. It turns out that there is a trade-off between the error rate and the computation time. We further formulate an optimization problem and propose a way to find the optimal design parameters.

The proposed accelerating module is applied to a stochastic implementation of a kernel density estimation-based image segmentation algorithm [9]. Experimental results showed that compared with the original system proposed in [9], the system with the accelerating module achieves $4.86\times$ speedup with area overhead less than 1.01% and error rate less than 0.4%.

II. The Accelerating Module

In this section, we show the accelerating module. It consists of a pre-processing block, which transforms the comparison threshold to the value 0.5, and a main block, which accelerates the comparison of a probability against the threshold 0.5.

A. Threshold Conversion Block

The accelerating module has a pre-processing part that converts the threshold to be compared against from an arbitrary value x to the value 0.5. The circuit is shown in Fig. 2. Its logic function is $C = (S \wedge A) \vee (\overline{S} \wedge \overline{B})$, where $\wedge$ and $\vee$ represent logical AND and OR, respectively.

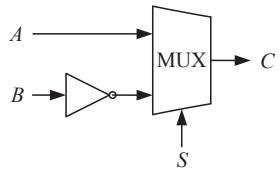

Fig. 2: A circuit for threshold conversion in stochastic computation.

If A, B, C, and S are stochastic bit streams and P_A, P_B, P_C, and P_S are their corresponding probabilities of ones, we have $P_C = P_S \cdot P_A + (1 - P_S) \cdot (1 - P_B)$.

If we want to compare against the threshold x, then we let S be a stochastic bit stream with probability 0.5 to have a one and B be a stochastic stream with probability x to have a one. Then we have

$$P_C = 0.5 \cdot (P_A + 1 - x) = 0.5 + 0.5 \cdot (P_A - x).$$

Based on the above equation, we can see that in order to check whether the probability of a one in A is larger than a threshold x, we only need to check whether the probability of a one in C is larger than 0.5.

B. Main Block of the Accelerating Module

The main block of the accelerating module checks whether the probability of a one in the entire stream is above or below 0.5. It splits the entire stochastic bit stream into d non-overlapping and consecutive segments of length n, where $d = \lfloor \frac{l}{n} \rfloor$. It checks these d segments one by one. To increase the confidence, a threshold $m > \frac{n}{2}$ is chosen. If in one segment, the number of ones (zeros) is at least m, then the module will infer that the probability of a one in the entire bit stream is above (below) 0.5 and the whole stochastic computing system will stop. If the module cannot make any inference from the current segment, it will check the next segment. If the module has examined all the d segments but fails to make any inference, it will derive the result based on the ratio of ones in the entire bit stream.

In order to further reduce the computation time, we apply an early termination strategy. In checking each segment, the module does not have to check all the n bits to make an inference. If the module finds that the number of ones (zeros) has reached m, it can immediately stop and infer that the probability of a one in the entire bit stream is above (below) 0.5.

C. Error Rate

In this section, we analyze the error rate of a stochastic computing system that uses the accelerating module. Assume the length of the entire stochastic bit stream is l and the stream is $X_1, X_2, \ldots, X_l$. Furthermore, assume that the sequence is a Bernoulli sequence, i.e., X_i's are independent and identical Bernoulli random variables. Assume $P(X_i = 1) = p$, for all $i = 1, \ldots, l$.

We focus on the error due to a wrong inference from *any* segment. First, we consider the case where the true probability $p > 0.5$. The module makes a wrong inference, i.e., $p < 0.5$, based on one of the segments if and only if there are at least m zeros in that segment, which is equivalent to the situation that there are at most $n - m$ ones in the segment. Therefore, the probability that a wrong inference is made based on a segment is

$$p_e = \sum_{i=0}^{n-m} \binom{n}{i} p^i (1-p)^{n-i}. \tag{1}$$

The module fails to make any inference from a segment if and only if the number of ones in the segment is between $n - m + 1$ and $m - 1$. Its probability is

$$p_f = \sum_{i=n-m+1}^{m-1} \binom{n}{i} p^i (1-p)^{n-i}. \tag{2}$$

Now we consider the event that the wrong inference is made from the i-th segment ($1 \leq i \leq d$). In this case, the module fails to make any inference from the previous $i - 1$ segments. Therefore, the probability that the wrong inference is made from the i-th segment is $p_e p_f^{i-1}$. Given this, the probability that the module makes a wrong inference from *any* segment is

$$p_e \cdot \sum_{i=0}^{d-1} p_f^i. \tag{3}$$

It can be shown that when $0.5 \leq p \leq 1$, Eq. (3) decreases with p. Therefore, it attains maximum when $p = 0.5$. Substituting $p = 0.5$ into Eq. (3) and simplifying it, we can obtain the maximum as

$$e_{\max} = \frac{1 - \left(1 - 0.5^{n-1} \sum_{i=0}^{n-m} \binom{n}{i}\right)^d}{2}. \tag{4}$$

We can similarly obtain the probability that the module makes a wrong inference from *any* segment when the true probability $p < 0.5$. It turns out that when $0 \leq p \leq 0.5$, that probability attains maximum when $p = 0.5$ and the maximum is of the same form as Eq. (4). Thus, for any $0 \leq p \leq 1$, the maximal error rate is given by Eq. (4), which is determined by the segment length n and the threshold m.

D. Computation Time

In this section, we analyze the computation time of a system that uses the accelerating module. The computation time is proportional to the number of bits T checked. Thus, we take T as a measure of the computation time. Since T is a random variable, we will analyze the *mean* computation time $E[T]$. In order to obtain $E[T]$, we will first obtain $E[T|p]$, the mean computation time given that the probability of a one in the stream is p. For this purpose, we first analyze the probability distribution of the random variable T under the assumption that the probability of a one in the stream is p.

Based on the behavior of the module, it is not hard to see that the possible values of T are of the form $(i-1)n + j$ ($1 \leq i \leq d$ and $m \leq j \leq n$) or l.

First consider the event that $T = (i-1)n + j$, for any $1 \leq i \leq d$ and $m \leq j \leq n$. This event means that the module stops computation at the j-th bit in the i-th segment. It occurs if and only if both of the following two events occur:

Event 1: The module fails to make any inference from the previous $i - 1$ segments.

Event 2: There are at least m ones or zeros in the i-th segment and the earliest time that the number of ones or zeros in the i-th segment reaches m is the j-th bit of the i-th segment.

The probability that Event 1 occurs is p_f^{i-1}. Now we consider when Event 2 happens. It happens either 1) $X_{(i-1)n+j} = 1$ and there are exactly $(m-1)$ ones in the first $(j-1)$ bits of the i-th segment, or 2) $X_{(i-1)n+j} = 0$ and there are exactly $(m-1)$ zeros in the first $(j-1)$ bits of the i-th segment. Therefore, the probability that Event 2 occurs (denoted as $P_{\text{seg}}(j)$) is

$$\begin{aligned} P_{\text{seg}}(j) &= p \cdot \binom{j-1}{m-1} p^{m-1}(1-p)^{j-m} \\ &\quad + (1-p) \cdot \binom{j-1}{m-1} (1-p)^{m-1} p^{j-m} \\ &= \binom{j-1}{m-1} \left[p^m (1-p)^{j-m} + (1-p)^m p^{j-m} \right]. \end{aligned}$$

Since all the bits in the stochastic bit stream are independent, we have the probability $P(T = (i-1)n + j) = p_f^{i-1} P_{\text{seg}}(j)$.

Now consider the event that $T = l$. It occurs only when the module cannot infer from any of the d segments. Then, it has to check the entire bit stream to obtain a result. The probability is $P(T = l) = p_f^d$.

Therefore, the conditional mean computation time of the system is

$$E[T|p] = l p_f^d + \sum_{i=1}^{d} \sum_{j=m}^{n} ((i-1)n + j) p_f^{i-1} P_{\text{seg}}(j).$$

Simplifying the above equation, we finally obtain

$$E[T|p] = (np_f + A + B) \frac{1 - p_f^d}{1 - p_f} + (l - nd) p_f^d, \tag{5}$$

where

$$\begin{aligned} A &= \frac{m}{p} \cdot P_B(X \le n - m | n+1, 1-p), \\ B &= \frac{m}{1-p} \cdot P_B(X \le n - m | n+1, p), \end{aligned}$$

and $P_B(X \le k|n, p)$ denotes the CDF of the binomial distribution.

Given $E[T|p]$ and the probability density function of the probability p, $f(p)$, we can obtain the mean computation time $E[T]$ as

$$E[T] = E[E[T|p]] = \int_0^1 E[T|p] f(p) dp. \tag{6}$$

E. Parameter Selection

Two important design parameters of the accelerating module are the segment length n and the threshold m. Given n and the length l of the entire bit stream, the number of segments d can be determined as $d = \lfloor \frac{l}{n} \rfloor$. Thus, d is not treated as a design parameter.

According to Eq. (4) and (5), both n and m affect the error rate and the mean computation time. A design goal of the proposed module is to minimize the mean computation time while making the error rate as low as possible. However, there is a trade-off between the error rate and the mean computation time: decreasing one increases the other. In order to find the optimal parameters n and m, we formulate an optimization problem, which takes the maximal error rate, given by Eq. (4), as a constraint and minimizes the mean computation time, given by Eq. (6):

> Given the length of the entire stochastic bit stream, l, and a limit on the error rate, ϵ, find parameters n and m ($\frac{n}{2} < m \le n \le l$) that minimize the mean computation time $E[T]$, while satisfying the constraint that the maximum error rate $e_{\max} \le \epsilon$. Here $E[T]$ and $e_{\max}$ are given by Eq. (6) and (4), respectively.

To solve the above optimization problem, we first consider when n is fixed, which m will minimize $E[T]$ while satisfying that $e_{\max} \le \epsilon$. It is not hard to see that when n is fixed, $e_{\max}$ given by Eq. (4) decreases with m for $\frac{n}{2} < m \le n$. Therefore, given the constraint $e_{\max} \le \epsilon$, there exists a minimal m that satisfies the constraint. On the other hand, we can prove the following theorem (due to the space limit, we omit the proof):

Theorem 1
When n is fixed, $E[T]$ given by Eq. (6) increases with m for $\frac{n}{2} < m \le n$. □

Thus, for a fixed n, the m that minimizes $E[T]$ is the minimal m that satisfies the constraint $e_{\max} \le \epsilon$, which we can obtain easily. In order to find the minimal $E[T]$ over all valid combinations of m and n, we iterate over all $1 \le n \le l$ to obtain l minima $E[T]$'s, each corresponding to a fixed n. The global minimum is the smallest one among these l minima.

III. Experimental Results

In this section, we show the experiment results on the proposed accelerating module. We first show the performance of the accelerating module. Then, as a case study, we apply the module to a stochastic implementation of an image segmentation application and show its real performance.

A. The Speedup with the Accelerating Module

We first studied how the speedup ratios due to the proposed module change with different bit stream lengths l. Without the module, we need to count all the l bits to obtain the result. However, with the module, we can reduce the mean computation time to the minimal value $E[T]^*$ using the optimal n and m. The speedup ratio is calculated as $l/E[T]^*$. We considered different bit stream lengths $l = 128, 256, 512, \ldots, 8192$. We assumed that the error rate limit is $\epsilon = 0.01$. Furthermore, we assumed that $f(p)$ in Eq. (6) is a uniform distribution in $[0, 1]$. The minimal mean computation time $E[T]^*$ and the speedup ratio due to the accelerating module for different l's are shown in Table I. From the table, we can see that the speedup ratio increases with l.

TABLE I: The minimal mean computation time $E[T]^*$ and the speedup ratio due to the proposed accelerating module for different bit stream lengths l.

l	128	256	512	1024	2048	4096	8192
$E[T]^*$	67.4	108.9	173.8	273.3	429.8	673.2	1053
Speedup	1.90	2.35	2.95	3.75	4.76	6.08	7.78

B. Case Study: Accelerating an Stochastic Implementation of an Image Segmentation Application

In this section, we take the stochastic implementation of the kernel density estimation (KDE)-based image segmentation algorithm as an example to demonstrate the effectiveness of the proposed module in accelerating the computation. KDE-based image segmentation applies statistical modeling to separate the foreground and background of an image [6]. It computes a probability density function $P(x)$ of the pixel's intensity x based on the previous samples. The pixel is considered to be a foreground pixel if $P(x)$ is smaller than a threshold z; otherwise, the pixel is considered to be a background pixel.

Li et al. proposed a stochastic implementation of the KDE-based image segmentation algorithm [9]. Compared with the conventional

implementation using binary radix encoding, the stochastic implementation only takes a very small circuit area. However, the stochastic implementation requires a long computation time.

In this experiment, we applied the proposed module to accelerate the stochastic implementation of the KDE-based image segmentation algorithm. We refer to the stochastic implementation proposed in [9] as the basic system. The accelerating module takes the output stochastic bit stream of the basic system, which encodes the value $P(x)$, as input. Once the module has determined whether $P(x)$ is larger or smaller than the threshold z, it resets the basic system so that the next pixel will be processed. We used Xilinx ISE Design Suite [13] for hardware prototyping.

Same as [9], we used a stochastic bit stream of 1024 bits to represent each value in the stochastic computation. We chose the limit on the error rate as $\epsilon = 0.01$. Without knowing the distribution of the $P(x)$'s for all the pixels in the image, we assumed that they are uniformly distributed. The optimal parameters n and m were obtained by solving the optimization problem. They are $n = 104$ and $m = 68$.

We tested the performance of the stochastic implementation with the accelerating module using a video sequence. The experimental results showed that the average number of bits that are checked to process a pixel in the image is 210.5. Since the basic system needs to check 1024 bits, the speedup ratio is 4.86. If we use the image segmentation result generated by the conventional KDE-based algorithm as the golden standard, then there are only 0.347% pixels that are different. Therefore, using the proposed module, the computation time is dramatically decreased, while the error rate is very low.

To visually demonstrate the quality of the result generated by the accelerated stochastic implementations, we show the output images generated by the conventional KDE-based algorithm and by the stochastic implementation with the accelerating module in Fig. 3 (b) and (c), respectively. We can see that the image generated by the accelerated stochastic implementation is close to the image generated by the conventional algorithm.

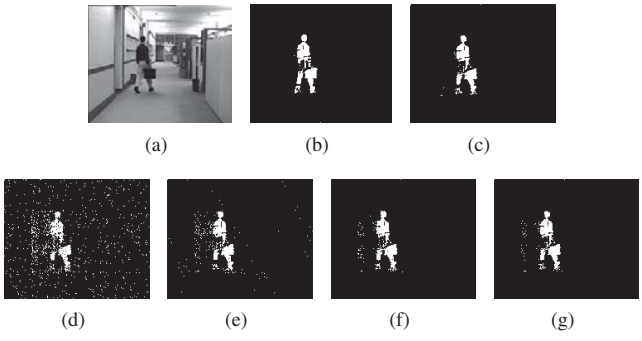

Fig. 3: Output images generated by the different implementations: (a) original image; (b) image by the conventional algorithm; (c) image by the accelerated stochastic implementation; (d) image determined by the first 50 bits. (e) image determined by the first 150 bits. (f) image determined by the first 250 bits. (g) image determined by the first 350 bits.

To further demonstrate the effectiveness of the proposed technique, we compared the stochastic implementation that uses the accelerating module to a simple acceleration strategy, which determines the result just based on the first k bits of the entire input bit stream. We applied the simple strategy to the same video sequence as we used above. Table II lists the speedup ratios and the error rates for $k = 50, 150, 250, 350$. In comparison, we also list the speedup ratio and the error rate of the stochastic implementation that uses the proposed module. From the table, we can see that the implementation using the proposed module has the lowest error rate. Furthermore, both the speedup ratio and the error rate of the implementation using the proposed module are better than those using the simple strategy for $k = 250$ and 350. Fig. 3 (c)–(g) further compare the output images of the implementation that uses the proposed module to those generated by the simple acceleration strategy for $k = 50, 150, 250, 350$. From those figures, we can see that due to our sophisticated accelerating approach, the output image is much clearer than those generated by the simple acceleration strategy.

TABLE II: Performance comparison between the stochastic implementation using the proposed module and those using a simple acceleration strategy that determines the result based on the first k bits of the entire bit stream.

	$k = 50$	$k = 150$	$k = 250$	$k = 350$	proposed
Speedup ratio	20.48	6.83	4.10	2.93	4.86
Error rate (%)	3.85	1.52	0.71	0.50	0.347

Finally, we studied the area overhead of the proposed module. We synthesized the design using Xilinx ISE Design Suite and obtained the resource usage from the synthesis report. Table III lists the resource usages of the basic system from [9] and the stochastic implementation with the proposed accelerating module. The table also lists the percentage of overhead, which is caused by including the accelerating module. From the table, we can see that the additional resource usage caused by the proposed module is negligible.

TABLE III: The hardware resource usage comparison.

	Usage of basic stoch. impl.	Usage of accelerated stoch. impl.	Overhead (%)
# Slices	6581	6631	0.76
# Slice Flip Flops	5640	5686	0.82
# LUTs	12230	12354	1.01

IV. Conclusion

In this work, we proposed a novel module to accelerate stochastic computation for binary classification applications. We analyzed the error rate and computation time of stochastic computing system that uses the proposed accelerating module. We also proposed how to find the best design parameters. We applied the proposed modules to accelerate the stochastic implementation of the KDE-based image segmentation algorithm. The experimental results showed that the accelerating module can help reduce the computation time dramatically with only a negligible area overhead and error rate.

Acknowledgement

This work is supported by National Natural Science Foundation of China (NSFC) under Grant No. 61204042 and 61472243.

References

[1] A. Alaghi, C. Li, and J. Hayes, "Stochastic circuits for real-time image-processing applications," in *Design Automation Conference*, 2013, pp. 1–6.

[2] C. Bishop, *Patten Recognition and Machine Learning*. Springer Publishing, 2007.

[3] S. Borkar, T. Karnik, and V. De, "Design and reliability challenges in nanometer technologies," in *Design Automation Conference*, 2004, p. 75.

[4] B. Brown and H. Card, "Stochastic neural computation I: Computational elements," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 891–905, 2001.
[5] J. Dickson, R. McLeod, and H. Card, "Stochastic arithmetic implementations of neural networks with in situ learning," in *International Conference on Neural Networks*, 1993, pp. 711–716.
[6] A. Elgammal, R. Duraiswami, D. Harwood, and L. Davis, "Background and foreground modeling using nonparametric kernel density estimation for visual surveillance," *Proceedings of the IEEE*, vol. 90, no. 7, pp. 1151–1163, 2002.
[7] B. Gaines, "Stochastic computing systems," in *Advances in Information Systems Science*. Plenum, 1969, vol. 2, ch. 2, pp. 37–172.
[8] R. Gonzalez and R. Woods, *Digital Image Processing, 3rd Ed.* Prentice Hall, 2007.
[9] P. Li, D. Lilja, W. Qian, K. Bazargan, and M. Riedel, "Computation on stochastic bit streams: Digital image processing case studies," *IEEE Transactions on Very Large Scale Integrated (VLSI) Systems*, 2013.
[10] J. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital integrated circuits – A design perspective (2/e)*. Prentice Hall, 2003.
[11] S. Tehrani, S. Mannor, and W. Gross, "Fully parallel stochastic LDPC decoders," *IEEE Transactions on Signal Processing*, vol. 56, no. 11, pp. 5692–5703, 2008.
[12] S. Toral, J. Quero, and L. Franquelo, "Stochastic pulse coded arithmetic," in *International Symposium on Circuits and Systems*, vol. 1, 2000, pp. 599–602.
[13] *ISE Design Suite*, Xilinx, Inc.
[14] Q. Zhou and K. Mohanram, "Gate sizing to radiation harden combinational logic," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, pp. 155–166, 2006.