DPS: Dynamic Precision Scaling for Stochastic Computing-based Deep Neural Networks*

Hyeonuk Sim, Saken Kenzhegulov, and Jongeun Lee
School of Electrical and Computer Engineering, UNIST, Ulsan, 44919 South Korea
{detective,skenzhegulov,jlee}@unist.ac.kr

ABSTRACT

Stochastic computing (SC) is a promising technique with advantages such as low-cost, low-power, and error-resilience. However so far SC-based CNN (convolutional neural network) accelerators have been kept to relatively small CNNs only, primarily due to the inherent precision disadvantage of SC. At the same time, previous SC architectures do not exploit the dynamic precision capability, which can be crucial in providing efficiency as well as flexibility in SC-CNN implementations. In this paper we present a DPS (dynamic precision scaling) SC-CNN that is able to exploit dynamic precision with very low overhead, along with the design methodology for it. Our experimental results demonstrate that our DPS SC-CNN is highly efficient and accurate up to ImageNet-targeting CNNs, and show efficiency improvements over conventional digital designs ranging in 50~100% in operations-per-area depending on the DNN and the application scenario, while losing less than 1% in recognition accuracy.

1 INTRODUCTION

Stochastic computing (SC) uses bitstreams whose values are represented by the frequency of ones vs zeros. The probabilistic nature of bitstreams allows for very low-cost implementation of common operations such as multiplication. Thanks to high error resilience and low implementation cost, SC is seen as a promising approach to accelerating certain applications including deep neural networks (DNNs) [7, 9–11, 13–17].

Previous work on SC-based DNNs (more in Section 2) has shown competitive results against conventional digital designs in terms of both accuracy and efficiency. It seems that the retraining capability of DNNs helps cope with the approximating nature of stochastic computing [7]. Also a recent technique [15] provides a highly efficient method of eliminating SNG (Stochastic Number Generator) overhead for DNNs. However, the inherent precision disadvantage of SC—that SC requires exponentially longer bitstreams as precision increases—has so far kept SC from being competitive on larger DNNs such as AlexNet [8].

At the same time, as DNNs grow more complex and diverse, there is a need for a more reconfigurable hardware architecture so that various DNNs with different precision requirements can be supported with high efficiency. This is particularly useful for SC, where 1-bit saving could reduce computation latency by 50% as shown in Figure 1, suggesting a great potential for higher efficiency on DNNs with layers of different precision requirements.

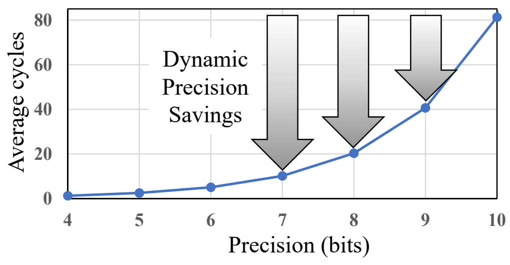


Figure 1: Dynamic precision, or using lower precision whenever possible, can give huge boost in efficiency for SC.

In this paper we present an SC-based CNN (convolutional neural network) accelerator that is not only highly efficient for large DNNs but also very flexible in terms of supporting various DNNs with different precision requirements. The key innovation in our proposed solution is to treat precision as a first-class design parameter and explicitly optimize its allocation across DNN layers and across value ranges of variables. Specifically, we first propose a DPS (Dynamic Precision Scaling) SC-MAC (multiply-and-accumulator), which extends a state-of-the-art SC-MAC [15] such that the precision of input/output data can be arbitrarily changed at runtime (see Section 3). The extension has very little overhead, but allows us to be efficiently parsimonious in terms of precision, with an exponential reward in latency due to SC. Second, in terms of allocating precision across value ranges of variables, we observe that in some layers input activations are always non-negative, meaning that we can reduce precision by 1-bit (and corresponding latency saving by up to 50%) and still get effectively the same accuracy. We call this optimization *Half-Range Specialization* (HRS). Importantly we implement the HRS optimization as an add-on feature that can be switched on dynamically, so that the same hardware can run all layers of a DNN regardless of the input range (see Section 3.3). Third, we present our design methodology to optimize DPS SC-DNNs in Section 4.

This paper presents a number of important results. First, we show for the first time that an SC-CNN can be as efficient as conventional digital designs up to ImageNet-targeting CNNs such as AlexNet [8] and VGG, with less than 1% degradation in recognition accuracy. This is quite significant as the previous work on SC-CNN was only able to show it up to Cifar-10, which is much smaller than ImageNet. Second, we show that our SC-CNN can be over 100% more efficient in terms of operations-per-area over conventional digital design when the same hardware is used for multiple CNN applications of varying precision requirements. Third, we show that even for a single application scenario where the hardware is designed only for one application (e.g., AlexNet), our SC-CNN can still be 52% more efficient than conventional digital design. These results demonstrate the flexibility and efficiency of our proposed solution.

*This work was supported by Samsung Research Funding Center of Samsung Electronics under Project Number SRFC-IT1501-08. *(Corresponding author: Jongeun Lee)*

DAC '18, June 24–29, 2018, San Francisco, CA, USA
© 2018 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.
ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3196028

2 RELATED WORK AND BACKGROUND

Previous work on SC-based acceleration of DNNs can be classified into two categories: fully-parallel and tile-based. In the fully parallel approach [7, 9–11, 13, 14], all neurons are implemented spatially and they operate in parallel such that the neural network circuit will produce output as the wave of input data sweeps through the circuit. It can have very high energy efficiency owing to the fact that it doesn't involve external memory access to store intermediate result, but has limited applicability because it cannot support arbitrarily large DNNs. The tile-based approach [15, 16] is more scalable in terms of the number of layers and neurons supported, since it works by tiling the computation of a layer into smaller fixed-sized arrays, each of which is performed by the same hardware block. The intermediate results are saved to and reloaded from buffers, which are typically on-chip, backed by external memories. For storage and bandwidth efficiency, SC data are often saved in memories as conventional digital numbers, which adds to the cost of implementation.

Recent CNN hardware implementations [1–3] all have their precision fixed at design time. This has obvious disadvantages when the precision requirement is different, such as low accuracy (when the needed precision is higher) and lower efficiency than achievable (when lower). Also since these CNN accelerators have tile-based architectures, all layers must use the same precision. However the precision requirement of layers can be quite different [6], which causes some inefficiency even when the accelerator is running the CNN for which it is designed.

One solution to these problems is to use bit-serial hardware such as bit-serial multiplier [6]. While this approach can solve the above mentioned problems, bit-serial multipliers are inherently inefficient compared with a bit-parallel version except in low precision, which limits the effectiveness of the approach.

In comparison, SC can be more efficient partially owing to the optimizations proposed in this paper. This helps close the efficiency gap between SC and bit-parallel conventional digital for a wider range of precisions as we show in Figure 5, while still retaining the benefits of SC such as dynamic precision scaling (DPS). Also the effect of DPS could be higher in SC than in conventional digital, since 1-bit reduction in SC may reduce area-delay product (ADP) by up to 50%.

Though SC multipliers are known to be very efficient in terms of ADP [7, 13], they come with a large SNG overhead, which is inevitable in tile-based architectures. But a recent work [15] integrates SNGs within an SC-MAC achieving orders of magnitude improvement in ADP. Part of this improvement is due to variable-latency multiplication, which is very effective for DNNs. Our contribution in this paper is to extend the SC-MAC to support DPS, together with an improvement and a design methodology for optimizing DPS SC-CNNs.

3 DYNAMIC PRECISION SCALING SC-MAC

3.1 Analysis of Baseline SC-MAC

Here we present an analysis of the SC-MAC proposed in [15]. The SC-MAC takes two operands labeled x and w, and generates output y which should approximate xw. All the inputs/output are represented as conventional digital, as illustrated in Figure 2 (shown in red is for dynamic precision discussed in the next section).

First consider the unsigned version, where the inputs/output are interpreted as fractional numbers between 0 and 1. Let $X = x2^Q$ and $W = w2^Q$, where Q is the width of the X and W registers. The

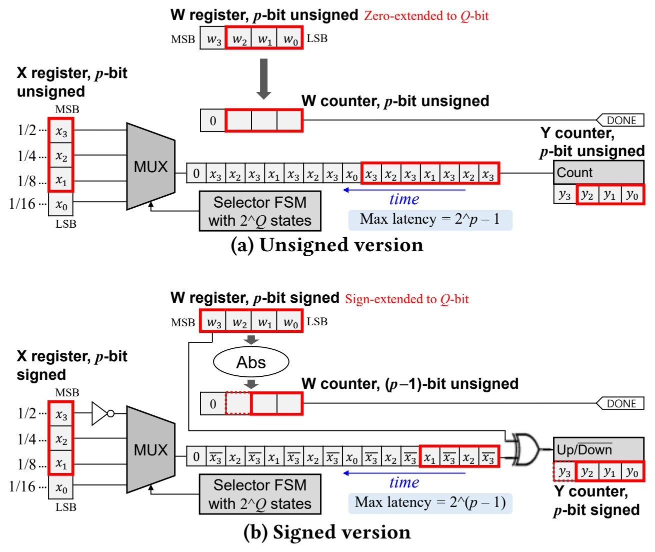


Figure 2: SC-MAC from [15] extended for dynamic precision. Datapath remains the same; only control is changed. (p: dynamic precision, Q: the maximum supported precision)

MUX-FSM circuit is designed to generate a bitstream whose signal probability is close to x [15]. Thus counting bits from the x-bitstream for W cycles gives approximately $xW = xw2^Q$. Therefore, $y \approx xw$ with Q-bit precision.

In the signed version, the inputs/output are 2's complement numbers between −1 and 1. Since MSB is used as the sign bit, $X = x2^{(Q-1)}$ and $W = w2^{(Q-1)}$. The W counter is initialized to the absolute value of W. If W is positive, feeding the x-bitstream to the Y counter (which is now an up/down counter) for W cycles will give approximately $xW = xw2^{(Q-1)}$.[1] If W is negative, the x-bitstream is inverted, resulting in the negated value of $x(-W)$, or xW in the Y counter. In either case, $y \approx xw$ with Q-bit precision (including the sign bit).

3.2 Extension to DPS SC-MAC

The extension to support dynamic precision scaling (DPS) on the SC-MAC level costs very little hardware. In fact, the datapath remains almost the same, the major difference being in the control logic. Here we explain the operation of DPS SC-MAC.

The DPS SC-MAC has an additional input p, which is the precision of the input/output values. Figure 2 illustrates an example with $p = 3$ where the maximum precision Q is 4-bit.

Let us first consider the unsigned version. The X register holds the integer version of x, or $x2^p$, aligned at the MSB. The remaining bits, if any, are not used. The W register is initialized to the integer version of w, or $w2^p$, zero-extended to fill the Q-bit register, i.e., $W = w2^p$. The selector FSM is unchanged regardless of p.

It is easy to see that the latency of multiplication is $W = w2^p$ cycles, which is at most $2^p - 1$. During this period, the LSB part of X register that is not initialized from x, is not used (x_0 in the example). Thus counting the x-bitstream still approximates xW, with less accuracy due to reduced precision of W. Since $Y \approx xW = xw2^p$, $y \approx xw$ with p-bit precision.

[1]To understand why: (i) Unsigned interpretation of X register after inverting the MSB is $(X + 2^{(Q-1)})/2^Q$. (ii) The expected contribution of a single bit z to an up/down counter is $(2z - 1)$. Thus the expected change of Y per cycle is $2(X + 2^{(Q-1)})/2^Q - 1 = x$.

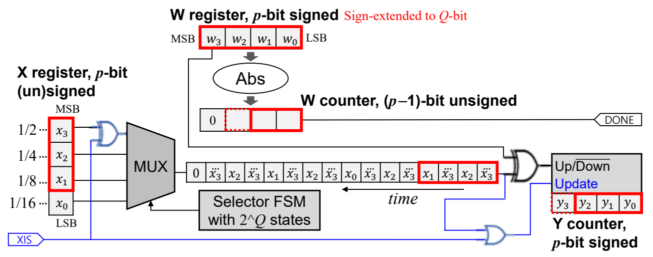

W register, p-bit signed
Sign-extended to Q-bit
MSB
w_3 w_2 w_1 w_0
LSB
Abs
X register, p-bit (un)signed
W counter, $(p-1)$-bit unsigned
0
DONE
MSB
1/2 x_3
1/4 x_2
1/8 x_1
1/16 x_0
LSB
MUX
0 $\ddot{x}_3$ x_2 $\ddot{x}_3$ x_1 $\ddot{x}_3$ x_2 $\ddot{x}_3$ x_0 $\ddot{x}_3$ x_2 $\ddot{x}_3$ x_1 $\ddot{x}_3$ x_2 $\ddot{x}_3$
time
Selector FSM with 2^Q states
Up/$\overline{\text{Down}}$
Update
y_3 y_2 y_1 y_0
Y counter, p-bit signed
XIS

Figure 3: Supporting HRS (Half-Range Specialization) mode for x along with the normal mode. Extension is shown in blue. The new input XIS indicates whether x is signed.

In the signed version, the X register holds $x2^{(p-1)}$ aligned at MSB. After inverting the MSB, the unsigned interpretation of X register is $0.5 + x/2$ with p-bit precision, assuming the decimal point right before MSB. The W register is initialized to $w2^{(p-1)}$, sign-extended to Q-bit, i.e., $W = w2^{(p-1)}$. If W is positive, the Y counter holds $xW = xw2^{(p-1)}$ after W cycles. If W is negative, it holds $-x(-W) = xW$ due to the XOR gate. In either case, $y \approx xw$ with p-bit precision including the sign bit.

The latency of signed multiplication is $|w|2^{(p-1)}$ cycles. The maximum latency is $2^{(p-1)}$ when $w = -1$, in which case the W counter requires p-bit, as indicated by the dotted box in Figure 2b (but it never needs more than Q bits). Similarly the Y counter needs $p + 1$ bits, which may exceed Q bits; however, the Y counter has extra bits already in order to serve as an accumulator.

Since the p-bit precision of y always starts from LSB, it means that the decimal point will have to move depending on the precision. Fixing the decimal point can be done with a single shifter.

3.3 Half-Range Specialization for DPS SC-MAC

Since the effect of precision in SC is exponential, saving even 1 bit is worthwhile. Half-range specialization (HRS) is based on the observation that the range of certain variables, namely, input activations, are guaranteed to be non-negative due to the particular shape of activation function (ReLU) used in the preceding layer.

One way to exploit the limited range of input is through a data scaling framework as in Section 4.3. But data scaling works best for symmetrical ranges, and making asymmetrical ranges symmetrical incurs additional overhead.

Alternatively we can make a full use of input precision in our DPS SC-MAC of Figure 2b by treating x as unsigned. This effectively increases x's precision to p-bit while the precision of w remains the same (i.e., $(p-1)$-bit due to 1-bit sign). Since w is unaffected, latency is also the same. The main effect of this scheme is accuracy improvement: in our evaluation, $(p-1)$-bit multiplication with HRS shows a similar accuracy as p-bit multiplication without HRS (see Figure 8). Conversely, HRS can achieve a similar accuracy with 1-bit less precision, or at half the latency.

It is important to note that HRS cannot guarantee the same accuracy as that of 1-bit lower precision, since w's precision is not increased. However, in most layers and most DNNs, input activations turn out to require a higher precision than weight parameters [5], which explains why increasing x's precision through HRS is very effective in practice.

Also important to note is that in order to support layers whose input is not necessarily one-sided (e.g., the first layer), the hardware must retain the original behavior of Figure 2b. Thus we make HRS runtime-programmable through an extra input XIS (meaning "x is signed"), as shown in Figure 3.

When XIS is 1, the hardware degenerates into the signed version. When XIS is 0, the x part becomes like the unsigned version, and the up/down operation of the Y counter is suppressed if the x-bitstream's output is 0, essentially making it perform either up or down, depending on the sign of w, which ensures a correct operation.

4 DESIGN OPTIMIZATION FOR DPS SC-CNN

4.1 Hardware Precision vs. Software Precision

So far our notion of precision has been the width of a variable in the application program as represented in conventional digital, which typically ranges up to 32-bit. Quantization is to reduce the precision in the application code. Thus this kind of precision may be called *software precision*. When converted into SC, a variable of n-bit software precision requires about a 2^n-bit bitstream.

While a bit-serial multiplier as in [6] computes only one bit at a time, it is quite common in SC to employ bit-parallel hardware. To generate a variable of n-bit software precision, a bit-serial SC multiplier needs 2^n cycles, but a k-bit parallel SC multiplier can do it in $2^n/k$ cycles. We define *hardware precision* as the base-2 logarithm of k.

We have extended our DPS SC-MAC to a bit-parallel version, which supports integer hardware precisions for efficiency reasons. It is based on the bit-parallel version of the baseline SC-MAC [15].

4.2 Design Flow

Our design objective is to minimize ADP while meeting accuracy constraint, which we set to be 1% point below the reference accuracy achieved by an unquantized version (i.e., floating-point implementation). We consider the following design parameters: (i) data scaling parameters, (ii) software precision of each layer, and (iii) hardware precision of SC-MAC. Next we discuss each of these.

4.3 Determining Data Scaling Parameters

Previous work [12] has pointed out the importance of scaling input data to better utilize the limited range of SC or fixed-point representations. The idea is to scale more than covering the worst case input data, such that some of the input values go out of range. It may introduce errors to some input, but those in the range can be represented more precisely. More-than-worst-case scaling is particularly effective when the out-of-range input data get saturated. To avoid the overhead due to scaling, scaling parameters are typically restricted to powers of 2.

The issue here is how to determine data scaling parameters, the effect of which seems highly unpredictable. We use the following scheme.

(1) Determine the scaling factor so that all values are within range (i.e., worst-case design).
(2) Double the scaling factor and check whether the recognition accuracy improves.
(3) Repeat the above while there is improvement.

The above procedure is repeated for each layer, starting from the first layer. We do not retrain the DNN during this procedure. We find this scheme robust as it does not rely on any arbitrary design parameter, which is a major advantage of the scheme. While this algorithm is greedy and not able to address the possible inter-dependence issue among layers, doing so would run into a combinatorial problem which can require prohibitive amount of resource for large DNNs.

4.4 Determining Software Precision of Each Layer

Similar to data scaling parameter exploration, here we optimize one layer at a time in order to avoid combinatorial problems. There are also differences. First, precision optimization uses retraining, which is crucial to get meaningful accuracy at low precisions. On the other hand, retraining takes much longer than inference, and can take hours and days for SC even when using GP-GPUs for simulation. Second, higher precision is more detrimental than a lower precision can save. Thus we first find the *uniform* precision for the SC version that satisfies the accuracy constraint with retraining. This can be solved in linear time, since all layers have the same precision. The uniform precision is used as the precision upper-bound for each layer. Third, knowing the uniform precision also helps determine hardware precision (see the next section). Fourth, to speed up the search we use the result of conventional digital implementation's optimized precision. However since there is usually a gap between the precisions of the two, we use a concept called *precision slack*.

To illustrate *precision slack*, suppose that a conventional digital implementation is optimized to have the following precisions across 5 layers: 10-9-5-6-8. Precision slack is the difference in precision between the highest and the current layer, e.g., 0-1-5-4-2 in this example. Then we subtract precision slack from the uniform precision value to get the precision lower-bound. The rationale is that while SC gives higher reward for lower precision, its accuracy is also more sensitive to it. Also using very low precisions gives diminishing return (see Figure 1) while often hurting accuracy too much. Having a lower-bound enables a binary search instead of a linear search, saving retraining time.

4.5 Determining Hardware Precision

Hardware precision affects both delay and area of our DPS SC-CNN (see Section 5.3). Therefore the decision in this step could affect the optimality of the precision setting found in the previous step as ADP can change as we use a different hardware precision. To avoid this problem, we run this step twice, first for the uniform precision value, then after non-uniform precision setting is found. Finding the best hardware precision is straightforward, and can be done quickly as it has only a linear complexity and doesn't require retraining. (Changing hardware precision doesn't affect recognition accuracy.)

5 EXPERIMENTS

5.1 Experimental Setup

To evaluate our approach we use ImageNet-targeting DNNs (AlexNet, VGG, GoogLeNet) in addition to smaller ones for comparison with previous work. For training and accuracy evaluation we use Caffe [4] extended for SC. Recognition accuracy is reported for the first 10,000 images of the ImageNet validation set. SC architecture is modeled cycle-accurately to generate exact cycle count in a data-dependent manner.

We have extended the SC-MVM (Matrix-Vector Multiplier) in [15] to use our DPS SC-MACs, which is referred to as DPS SC-MVM. The previous SC-MVM, our DPS SC-MVM, and the conventional digital baseline MVM are all implemented in Verilog and synthesized

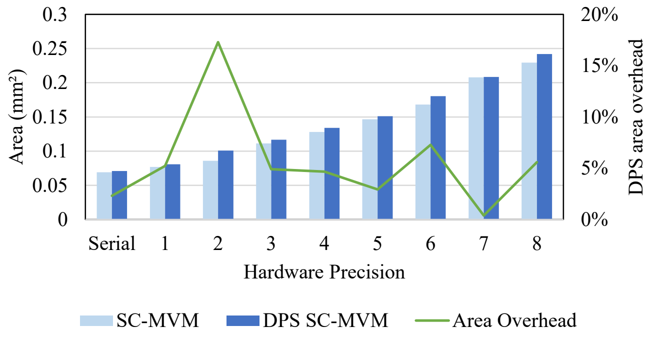


Figure 4: Area overhead of DPS-MVM

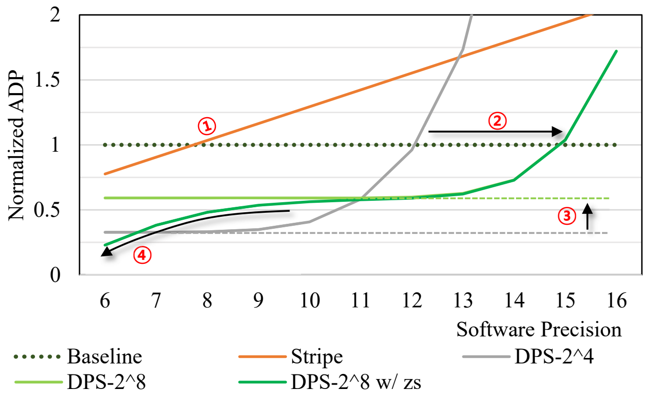


Figure 5: ADP vs. software precision.

using Synopsys Design Compiler. All syntheses are done for the same target frequency, 1 GHz, although SC is more likely to meet higher frequency. The conventional digital baseline uses fixed-point binary multipliers with rounding accumulators. The area for Stripe [6] is estimated to be 207% of the digital baseline as per their paper. Only convolution layers are accelerated in all the approaches compared, permitting us to use ideal convolution layer speedup for delay comparison. Accuracy degradation is set to 1% point.

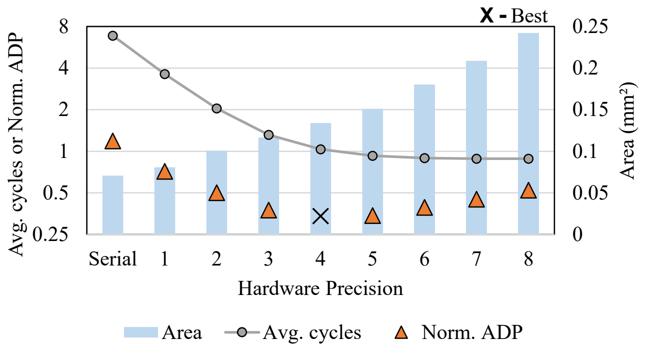
Our main figure of merit is area-delay product (ADP), which is the product of MVM area and average MAC cycles, or its inverse representing operations-per-area.

5.2 Area Overhead of Our DPS SC-CNN

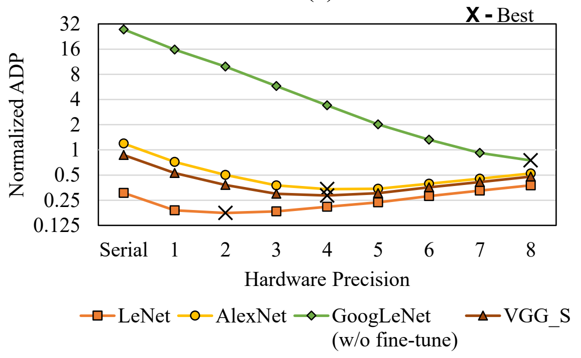
Figure 4 compares the area of our proposed DPS SC-MVM against the previous SC-MVM [15]. The DPS SC-MVM includes our optimizations such as HRS, which have very small extra logic (see Figure 3). Not surprisingly, the graph shows that the area overhead of ours is mostly small, typically at around 5%, though varied depending on the hardware precision shown on the x-axis. The graph also shows that the area is linearly proportional to the hardware precision, or logarithmically to bit-parallelism. This is due to the optimization exploiting the structure of the bitstream ordering. Overall the average area overhead is 6%, which is small.

5.3 ADP vs. Software and Hardware Precisions

Figure 5 shows the ADP trend as we vary software precision. For our ADP result we use AlexNet parameters as our SC-MAC has data-dependent variable latency. The digital baseline does not support



(a)



(b)

Figure 6: ADP vs. hardware precision.

dynamic precision, thus constant ADP. For stripe, delay is proportional to the precision, resulting in linear ADP. The graph shows that Stripe becomes inefficient over the conventional digital baseline beyond 7- or 8-bit (①). DPS-2ˆ4, which is our DPS SC-MAC with hardware precision of 4, confirms the exponential increase of ADP as software precision increases. But the range of software precision for which DPS-2ˆ4 is more efficient than conventional digital is wider than that of Stripe.

DPS-2ˆ8, which is our DPS SC-MAC with hardware precision of 8, can widen the efficient operating range even further (②). At the same time, it has higher ADP than DPS-2ˆ4 when software precision is lower (③), as it is more optimized for higher precision workload. Some of the efficiency loss can be reclaimed by zero skipping (④), which is to skip computation of multiplication whose w operand is 0 (after quantization) as shown in the graph.

As demonstrated, ADP depends on hardware precision. As hardware precision increases, (average) delay decreases monotonically until it saturates, whereas area increases more or less linearly to hardware precision. This leads to an optimization issue for hardware precision. Figure 6 shows ADP vs. hardware precision (a) for a single DNN and (b) for multiple DNNs.

The first graph is for AlexNet. As hardware precision increases, delay is reduced at first but eventually saturates. In this example DPS-2ˆ4 is the optimal. But in other CNNs, different points can be optimal as there are different weight distributions and precision requirements depending on the CNN. Figure 6b shows how ADP as well as optimal hardware precision changes depending on application. Understandably large and high precision CNNs seem to be better off with higher hardware precision. Our hardware precision

Table 1: Recognition accuracy (for 10K images) and DPS precision setting

CNN	float	DPS	DPS precisions found
MNIST	0.9904	0.9826	5 (uniform)
AlexNet (top-5)	0.807	0.7999	10-9-8-9-9
GoogLeNet (top-5)	0.8926	0.8844	13 (uniform, w/o fine-tune)
VGG_S (top-5)	0.8341	0.8247	9-9-10-9-10

for the multi-application scenario (see the next section) is chosen based on this profile.

5.4 Multi-application Scenario

Figure 7a compares our DPS SC-CNN and previous CNNs. First ours is highly area efficient which is not surprising given the area efficiency of SC. Stripe has the largest tile area because the number of MACs is 16 times greater than that of others, as shown in Figure 7b; the others have 256 MACs only.

Figure 7d shows the ADP result, which is normalized to the digital baseline. First, all these results are from implementations that achieve less than 1% point accuracy drop from the reference floating-point implementations (see Table 1). That SC-CNNs can achieve this high accuracy for large CNNs is very significant. Also this is why this graph has no comparison with previous SC-CNNs. Second, at the same time the efficiency of ours as measured in ADP is actually higher than conventional digital, often significantly. For instance, DPS-2ˆ8, which is optimized for large CNNs, shows consistently better results than previous digital designs. It also demonstrates the flexibility as well as efficiency of our DPS SC-CNN. Third, the optimal design as measured in geometric mean of ADP is DPS-2ˆ6 for this mix of CNNs, which is obviously influenced by the existence of a small network. But our scheme can flexibly support different workloads through the hardware precision, while simultaneously being able to support dynamic software precision at runtime. Overall, our DPS-2ˆ6 achieves more than 2X and 1.5X improvements over the baseline and Strip, respectively, in terms of operations-per-area.

5.5 Single Application Comparison

We also compare different CNNs including the previous state-of-the-art SC-CNN [15], when they are designed and used for just a single CNN, with AlexNet as the example. Figure 8 shows area, average delay, and ADP results in one graph, all normalized to that of the digital baseline. For SC designs, hardware precision is set to 4. Maximum software precision supported (Q) is determined to be the minimum value that meets the recognition accuracy constraint, which is largely dependent on how accurate the MAC is. The digital baseline requires 9-bit while the previous SC-CNN requires 11-bit. Our DPS SC-CNN requires 10-bit with uniform precision; dynamic precision setting is listed in Table 1.

The graph shows that previous SC-CNN has smaller area than the digital baseline but its average delay is much higher, which is due to the high precision requirement. Applying HRS to it (but not DPS) can reduce precision requirement by 1-bit with significant saving in delay, but its average delay is still higher than that of conventional digital. Applying DPS further gives 14% more reduction in ADP, achieving the best efficiency. The relatively weak impact of DPS is due to the small number of layers in AlexNet and limited precision exploration. For deeper networks and if optimal precision combinations can be used, the impact is likely higher. Even with these limitations our proposed design achieves 34% and 46% reduction

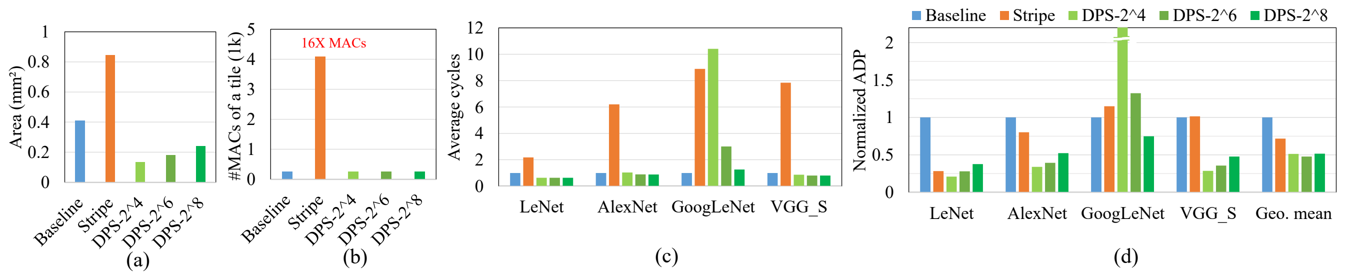


Figure 7: Comparison with the digital baseline and Stripe [6].

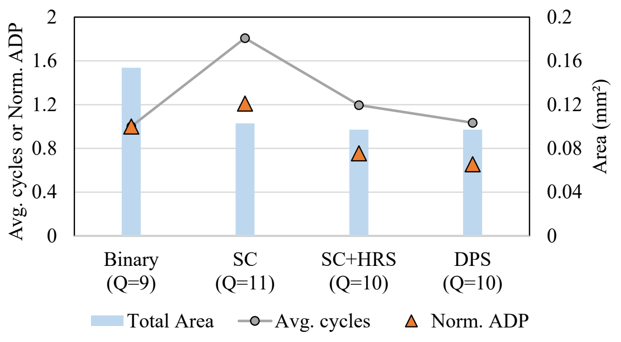


Figure 8: Comparison with previous SC-CNN on AlexNet.

Table 2: Feature comparison (O: supported)

	Digital		SC			
	[1]	[6]	[7]	[16]	[15]	Ours
Large (≥ 5 Conv. layers) DNNs	O	O	X	X	X	O
Tile based	O	O	X	O	O	O
Per-DNN precision	X	O	X	X	X	O
Per-layer precision	X	O	X	X	X	O
Per-bit precision	X	O	O	O	O	O
Multi-bit acceleration	X	X	X	X	O	O
Variable latency operation	X	X	O	X	O	O
Half-range specialization	X	X	X	X	X	O

in ADP (or 52% and 85% increase in operations-per-area) over the digital baseline and the previous SC-CNN, respectively.

5.6 Feature Comparison with Previous Work

In addition to performance numbers, our solution proposed in this paper has many important features as summarized in Table 2. One of the key factors that ours is able to give better efficiency than the previous work is the combination of variable latency and dynamic precision (per-layer precision), and multi-bit acceleration which is crucial to be efficient for larger CNNs, and HRS, which is specific to SC.

6 CONCLUSION

In this paper we presented a set of optimizations to enable highly accurate and efficient SC-based CNN implementations up to ImageNet-targeting CNNs. This is a significant leap for SC as the previous work has remained at much smaller datasets. Our key ingredient is the optimal use of precision—arguably the most scarce resource in SC—which is enabled by our DPS SC-MAC that can efficiently vary the input/output precision at runtime. Half-range specialization increases the effective precision by 1-bit without increasing the latency. DPS is also crucial in providing flexibility for SC-CNN implementations. The flexibility of DPS comes with the challenge of optimizing it. Currently our optimization flow is greedy and slow due to the retraining of SC-CNN. Finding near-optimal SC-CNN designs reliably and much more quickly remains for future work.

REFERENCES

[1] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and others. 2014. Dadiannao: A machine-learning supercomputer. In *MICRO'14*. IEEE Computer Society, 609–622.
[2] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138.
[3] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. In *ISCA'16*. IEEE Press, 243–254.
[4] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093* (2014).
[5] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, Raquel Urtasun, and Andreas Moshovos. 2015. Reduced-precision strategies for bounded memory in deep neural nets. *arXiv preprint arXiv:1511.05236* (2015).
[6] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M Aamodt, and Andreas Moshovos. 2016. Stripes: Bit-serial deep neural network computing. In *MICRO'16*. IEEE, 1–12.
[7] Kyounghoon Kim, Jungki Kim, Joonsang Yu, Jungwoo Seo, Jongeun Lee, and Kiyoung Choi. 2016. Dynamic Energy-accuracy Trade-off Using Stochastic Computing in Deep Neural Networks. In *DAC'16*. Article 124, 6 pages. DOI: https://doi.org/10.1145/2897937.2898011
[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS'12*. 1097–1105.
[9] Ji Li, Ao Ren, Zhe Li, Caiwen Ding, Bo Yuan, Qinru Qiu, and Yanzhi Wang. 2017. Towards acceleration of deep convolutional neural networks using stochastic computing. In *ASP-DAC'17*. 115–120.
[10] Zhe Li, Ao Ren, Ji Li, Qinru Qiu, Yanzhi Wang, and Bo Yuan. 2016. Dscnn: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks. In *ICCD'16*. 678–681.
[11] Zhe Li, Ao Ren, Ji Li, Qinru Qiu, Bo Yuan, Jeffrey Draper, and Yanzhi Wang. 2017. Structural design optimization for deep convolutional neural networks using stochastic computing. In *DATE'17*. 250–253.
[12] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. 2016. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*. 2849–2858.
[13] Ao Ren, Zhe Li, Caiwen Ding, Qinru Qiu, Yanzhi Wang, Ji Li, Xuehai Qian, and Bo Yuan. 2017. Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing. In *ASPLOS'17*. 405–418.
[14] Ao Ren, Zhe Li, Yanzhi Wang, Qinru Qiu, and Bo Yuan. 2016. Designing reconfigurable large-scale deep learning systems using stochastic computing. In *ICRC'16*. 1–7.
[15] Hyeonuk Sim and Jongeun Lee. 2017. A New Stochastic Computing Multiplier with Application to Deep Convolutional Neural Networks. In *DAC'17*. Article 29, 6 pages. DOI: https://doi.org/10.1145/3061639.3062290
[16] H. Sim, D. Nguyen, J. Lee, and K. Choi. 2017. Scalable stochastic-computing accelerator for convolutional neural networks. In *ASP-DAC'17*. 696–701. DOI: https://doi.org/10.1109/ASPDAC.2017.7858405
[17] Aidyn Zhakatayev, Sugil Lee, Hyeonuk Sim, and Jongeun Lee. 2018. Sign-Magnitude SC: Getting 10X Accuracy for Free in Stochastic Computing for Deep Neural Networks. In *DAC'18*. 6. DOI: https://doi.org/10.1145/3195970.3196113