AxBy: Approximate Computation Bypass for Data-Intensive Applications

Dongning Ma
Villanova University
Villanova, PA 19085, USA
dma2@villanova.edu

Xun Jiao
Villanova University
Villanova, PA 19085, USA
xjiao@villanova.edu

***Abstract*—Recent years have witnessed a rapid growth of data-intensive applications such as machine learning and multimedia applications. However, such applications incur a heavy computation workload that stresses the existing computing systems, especially resource-constrained embedded systems. This paper is inspired by the key observation that many data-intensive applications naturally present a strong existence of trivial computations — a set of computations the results of which can be determined without actual computations. Typical examples include multiplication with 0, +1/-1 and addition with 0. Correspondingly, we develop and implement bypass circuits that are tightly integrated with computation units to detect and bypass the trivial computations. Once detected, the circuit delivers the pre-determined result without an actual computation. We implement bypass circuits in both hardware (Verilog) and software (C). Furthermore, we enhance the opportunities of computation bypass by developing `AxBy`, an approximate computation bypass method with pattern matching under limited data precision. This reconfigurability is key to achieving a "controllable approximation" and a tunable quality-energy tradeoff. Our experimental results show that for four image processing applications and three neural network applications, the computation bypass can enable 15% – 55% in image processing and 30% – 35% in neural networks of energy saving without any accuracy loss. For neural networks, we can further achieve 36% – 44% energy saving with negligible accuracy loss.**

I. Introduction

Emerging data-intensive applications such as image processing [6], deep learning [2], and big data analysis [24] have significantly developed and had tremendous success. Recent trends to enhance and promote these applications has resulted in their implementations in various computing platforms including resource-constrained embedded systems. However, the increasingly intensive computing workload along with the growing complexity of these applications is creating increasing energy consumption demands that challenge hardware platforms. For example, even for a single input query, deep neural networks (DNNs) require billions of addition and multiplication operations [3].

To combat this challenge, recent approaches reduce the computing workload by using approximations in computed results, often referred to as "approximate computing". Voltage scaling is a widely-used approximation method. It reduces the supply voltage of hardware circuits, thus reducing the energy consumption of each individual circuit operation [22]. However, voltage scaling can cause timing errors in circuits which are hard to assess and control. This can compromise the output quality or even corrupt system reliability. Recent research shows that voltage scaling can cause significant accuracy loss in DNNs [16]. Another widely-used approximation method is the approximate computation reuse [7], [15]. This method is accomplished through the approximate reuse of computed results to reduce redundant executions. While effective, this technique has several disadvantages. First, it requires offline profiling to extract the most frequent computation patterns in an application, which is time-consuming and cost-prohibitive. Second, in order to implement energy-efficient associative memory, designers typically need to deploy advanced elements such as resistive memory elements [7], which increases the difficulty of practical deployment.

DNNs are naturally suited to such approximations because of their intrinsic error tolerance [5], [21], [25]. From the algorithmic perspective, pruning is used to compress and shrink the original dense model size to a sparse network structure [9], [10]. Quantization reduces the parameter precision by converting the floating point operations into fixed point operations [20], [8]. Hardware-level approximation includes substitution of exact multipliers with approximate multipliers [5], [21] and selective replacement of the less-critical neurons with approximate neurons [25]. While the adaptability makes neural network a natural target for approximation, in practice it also requires *retraining* or fine-tuning to mitigate approximation-induced errors, which can be cost-prohibitive and time-consuming.

To overcome the above-mentioned limitations, we propose **`AxBy`**, a retraining-free method for reducing the computation workload of DNNs by detecting and bypass the trivial computations. **`AxBy`** is inspired by the key observation that many data-intensive applications have a strong data locality and considerable number of *trivial computations*. The results of trivial computations can be pre-determined without performing the actual operation, i.e., without triggering the computation units. For example, for a multiplication, one type of trivial computation is when any operand equals to 1 because the result is simply the other operand. Thus, such computations can be bypassed. Correspondingly, we design bypass circuits that can detect such trivial computations. Once detected, the bypass circuits will return the pre-determined results.

We make the following contributions in this paper:

- At the software level, we explore the data locality and

the existence of trivial computations in image processing applications and neural network applications, then define and classify trivial computations into multiple categories.
- We leverage the inherent error tolerance of data-intensive applications and explore the use of approximation methods. We propose **AxBy**, an approximate computation bypass method that can match the trivial computations with reconfigurable data precision, i.e., different bit widths, to further enhance the bypass opportunity and improve the hit rate.
- At the hardware level, we implement bypass circuits that can detect and bypass trivial computations. We design and physically implement bypass circuits in Verilog and measure their post-layout energy/area overhead in TSMC 45nm technology.
- We evaluate the effectiveness of the proposed approach on four image processing applications and three neural network applications. Experimental results show that **AxBy** can enable 15% – 55% energy saving in image processing and 30% – 35% in neural networks without any loss in accuracy. For neural networks, we can further achieve 36% – 44% energy savings with negligible accuracy loss.

The remainder of this paper is organized as follows: Section II introduces necessary background on image processing applications and neural networks. Section III presents the overview of **AxBy**. Section IV describes the definition, categories and detection of trivial computations as well as the hardware design and implementation of **AxBy**. Section V demonstrates the effectiveness of **AxBy** on various benchmarks. Section VI presents related works. Section VII concludes this paper.

II. Background

A. Image Processing Application

We focus on four widely-used image processing applications in this paper: Sobel filter, Roberts filter, Scharr filter, and Sharpen filter. These applications share similar computation processes: two kernels which are convolved with the original image to calculate approximations of the derivatives. For example, as shown in Fig. 1, in calculating the gradient, the majority of computations are additions and multiplications. It is also worth noting that the kernel matrices are mainly composed of 1s and 0s, which motivates the use of data locality.

B. Convolutional Neural Networks

Modeled after brain-inspired biological neuronal processing, (artificial) neural networks are a family of problem-solving methods in machine learning. Recently, convolutional neural networks (CNNs) have been increasingly popular in various applications due to their superior accuracy [13]. Fig. 2 depicts LeNet-5 [19], a typical CNN architecture that consists of six layers, where the first, third, and fifth layer are convolutional, while the second and fourth are pooling layers, and the sixth is a fully connected layer.

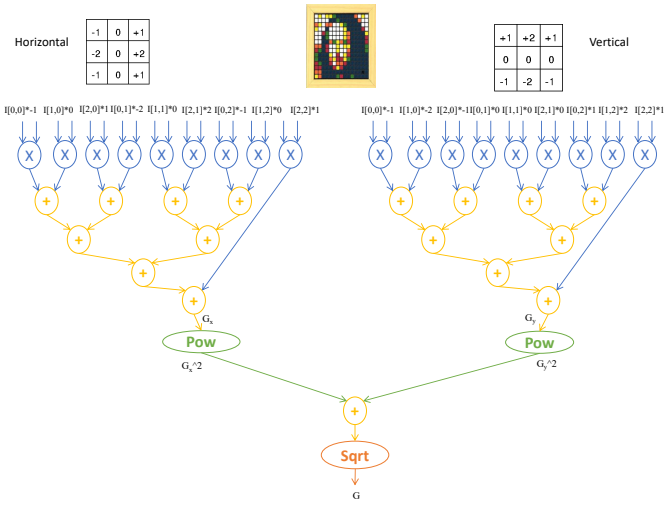


Fig. 1. Sobel filter data flow graph

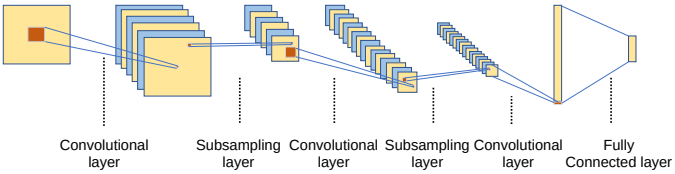


Fig. 2. An illustration of a convolutional neural network.

Convolutional layers which contain a number of filters are the most computation-intensive layers inside the CNN. These sliding filters are used to perform convolutions with a portion of the input image to generate an output image, namely the feature map. The basic unit in an artificial neural network is called the neuron, which performs the basic computations as illustrated in Fig. 3. The computation process of a typical neuron consists of linear processing followed by non-linear processing. The linear processing is a weight sum of the products of inputs and corresponding weights. The non-linear processing will use an activation function such as rectilinear unit (ReLU) to map the weighted sum to a specific range. Finally, the output y_k of neuron k is computed as $y_k = \delta(\sum_{j=1}^{n} x_j w_{jk} - \theta)$, where x_j is the j^{th} input, w_{jk} is the synaptic weight connecting the j^{th} input and neuron k, θ is the bias, and δ is the activation function. While fixed point implementations of neural networks are widely used in mobile platforms, floating point implementations are still preferred in

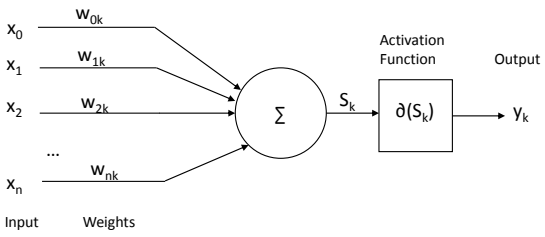


Fig. 3. The computation processes of an artificial neuron.

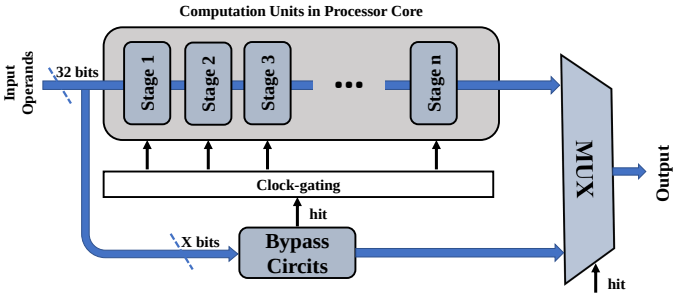
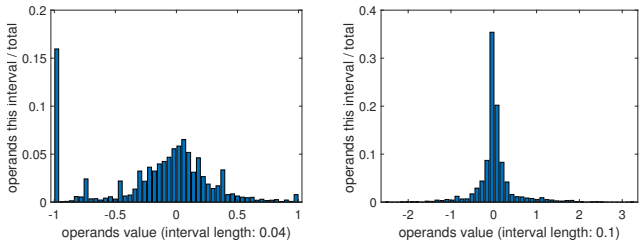


Fig. 4. Overview of **AxBy**.



(a) MUL (b) ADD

Fig. 5. Histograms of MUL and ADD operands from CNNs.

data centers due to accuracy requirements such as ResNet [11] and ResNeXt [27]. In this paper, we target CNNs with floating point operations, which are processed by floating point units (FPUs).

III. AxBy Overview

Fig. 4 presents an overview of **AxBy**. We tightly integrate the bypass circuits with computation units. For each computation, the two input operands are fed to both the computation unit and the bypass circuit. The bypass circuit will detect whether it is a trivial computation based on the pattern matching of input operands. If a trivial computation is detected, the bypass circuit will "clock-gate" the computation unit and send a "hit" signal to the multiplexer. Under the approximate mode, only the first "X" bits of input operands will be used to perform pattern matching and determine a "hit". The multiplexer will then select and output the predetermined result of trivial computations. If there is no trivial computation detected, the computation unit will execute normally and the multiplexer will output the computation result from the computation unit.

The advantages of **AxBy** are five-fold:

- It is retraining-free: when used in CNNs, it does not require retraining to reclaim the accuracy loss.
- It is profiling-free: it can be directly applied in hardware without the need to profile computation patterns offline.
- It offers controllable approximation: the number of bits used to match the trivial computations is controllable and hence can be used to control the approximation level.
- It has low-overhead: the bypass circuits are composed of a few logic gates and hence is low cost in terms of energy and area.
- It is platform-independent: the bypass circuits are tightly integrated with computation units that may be deployed in any platform such as data centers or edge devices.
- It is complimentary to existing methods such as pruning and quantization.

IV. AxBy Design

A. Case Study on Data Locality

We perform a case study on the data locality of CNN as an example of data-intensive applications. We use the MNIST database of handwritten digits as the dataset and LeNet-5 as our network architecture. To highlight the distribution of computation operands, we draw histograms of multiplication (MUL) and addition (ADD) operands from the convolutional layers as shown in Fig. 5. For MUL, as shown in Fig. 5(a), around 15% of input operands fall into the interval near -1 while the rest exhibit a Gaussian-like distribution with a center around 0. For ADD, per Fig. 5(b), more than 30% of the operands are centered around 0. Such observations indicate that the data locality of data-intensive applications is notable and the significant amount of clustered operands can be leveraged to enhance computation bypass via approximate pattern matching.

B. Trivial Computation Definition

The results of trivial computation can be determined without actual operation of the computation unit. Table I denotes the definition of trivial computations in ADD and MUL under different categories. According to the number of operands involved in determining the results of a trivial computation, the trivial computations from one type of operation are further classified into "full" or "semi" trivial. "Full" denotes that the result is not related to any of the operands while "semi" means the result is dependent on one operand. Intuitively, "full" trivial computations are prioritized over "semi" ones when both criteria are satisfied. This is because results from "full" trivial computations are usually "0" as listed in "Result" column in Table II and Table III, which satisfies the criteria of another trivial computation and triggers additional matches.

C. Trivial Computation Detection

According to Table I, there are three cases related to operands to be detected: "0" operands (MUL-Full and ADD-Semi), "$\pm$ 1" operands (Mull-Semi) and two inverse operands (ADD-Full). In this work, operands from trivial computations are floating point numbers complying with the IEEE-754 format for CNN applications and integers for image processing applications. Since these two data types follow different formats, we provide two tables, Table II and Table III, on how to use the bit-wise representation of operands to detect and classify the trivial computations.

Take 1.00×2.25 as an example:

1.00: 0 01111111 00000000000000000000000

2.25: 0 10000000 00100000000000000000000

Since 1 matches the bit representation listed in Table I, this operation will be detected as a "semi" trivial multiply computation. The ongoing processing by computation unit will

TABLE I
CATEGORIES OF TRIVIAL COMPUTATIONS IN **AxBy**

	Description	Expression	Result
MUL-Full	any of the operands equal to 0	$p = 0$ or $q = 0$	0
MUL-Semi	any of the operands equal to 1 or -1	$\|p\| = 1$ or $\|q\| = 1$	$\pm p$ or $\pm q$
ADD-Full	the two operands are inverse numbers of each other	$p + q = 0$	0
ADD-Semi	any of the operands equal to 0	$p = 0$ or $q = 0$	q or p

TABLE II
AxBy TRIVIAL COMPUTATION BIT REPRESENTATION: FLOAT

value		bit representation
"0"		X 00000000 00000000000000000000000
"±1"		X 01111111 00000000000000000000000
"inverse"	a	0(1) .....
	b	1(0) .....

X: "don't care" bit

TABLE III
AxBy TRIVIAL COMPUTATION BIT REPRESENTATION: INTEGER

value		bit representation
"0"		X 0000000 00000000 00000000 00000000
"±1"		X 0000000 00000000 00000000 00000001
"inverse"	a	0(1) .....
	b	1(0) .....

X: "don't care" bit

subsequently be "clock-gated", and the "hit" signal will control the multiplexer to return the result 2.25 (the other operand).

D. Exact bypass

If a trivial computation is successfully detected and bypassed, we call it a "hit". We then use "hit rate" to quantitatively measure the ratio of trivial computations to the total computations. Naturally, a higher hit rate indicates more trivial computations and higher potential for energy savings. We use several different applications from four image processing applications (Sobel, Scharr, Roberts and Sharpen filters) to CNNs under three different datasets (MNIST, EMNIST and CIFAR-10). Note that CNNs use floating point numbers while image processing applications use integer numbers.

E. Approximate Bypass

Exact bypass only catches the cases when all bits in the operands are matched. However, according to Fig. 5, there is a significant number of operands clustering around "trivial operands" such as "0" and "1", showing the potential of leveraging approximate bypass method. That is, instead of checking all bit positions to detect a "hit", approximate bypass only checks a limited number of bits for pattern matching. Specifically, if the first "X" bits of incoming operand matches trivial operand, it can be treated as a trivial computation.

Take 1.0625×2.25 as an example:

1.0625: 0 01111111 00010000000000000000000

1: 0 01111111 00000000000000000000000

2.25: 0 10000000 00100000000000000000000

With the exact bypass scheme, this computation will not be detected as trivial because the fourth mantissa bit of 1.0625 and 1 is different. However, under approximate bypass with 12-bit match setting, for example, only the first 12 bits will be checked and compared, rendering a successful "hit".

1.0625($\approx$ 1.00): 0 01111111 000

2.25: 0 10000000 001

Therefore, this computation will be regarded as trivial and there will be a hit. Note that the result of this computation will be returned as 2.25, so there will be a deviation between the exact computation result and the returned result: $|2.25 \times 1.0625 - 2.25 \times 1| = 0.140625$, which will subsequently introduce errors in the applications. Clearly, there is a tradeoff between accuracy and hit rate: the fewer bits we use in pattern matching, the higher the hit rate and accuracy loss.

To quantitatively measure the errors introduced by approximation, we define quality loss as follows. For CNNs, the accuracy loss is the degradation of accuracy from baseline implementation (without approximate bypass) after using approximate bypass. For image processing applications, we use peak signal-to-noise ratio (PSNR) as our metric. We deem images with $PSNR \geq 30dB$ as acceptable quality. The quality loss is the percentage of images that have $PSNR < 30dB$.

Algorithm 1 floating point multiplication bypass of **AxBy**

Given bit-accuracy $bacc$, operand 1 $opr1$, operand 2 $opr2$
Return output result res

$opr1t, opr2t \leftarrow Truncate(opr1, opr2, bacc)$
if $opr1t$ is 0 or $opr2t$ is 0 **then**
  return 0
else if $|opr1t|$ is 1 **then**
  return $\{sgn(opr1t)$ XOR $sgn(opr2t)$, $opr2t[30:0]\}$
else if $|opr2t|$ is 1 **then**
  return $\{sgn(opr1t)$ XOR $sgn(opr2t)$, $opr1t[30:0]\}$
else
  return $opr1 \times opr2$
end if

F. *AxBy* Implementation

We implement **AxBy** in both software (C) and hardware (Verilog). The software version is used to measure the hit rate of trivial computations while the hardware version is used to measure the overhead. As an example, we show how we can implement **AxBy** for floating point multiplication in Alg. 1. First, the operands ($opr1$ and $opr2$) will be truncated according to the bit match setting specified by the input argument. Then the truncated operands ($opr1t$ and $opr2t$) will be checked for

a match with the criteria of trivial computation. **AxBy** will check the two operands first since full-trivial conditions are prioritized over semi-trivial conditions as aforementioned. If any of them is equal to "0", the output of this computation will be returned as "0". If there is no match for full-trivial conditions, **AxBy** will check if the semi-trivial condition is met, i.e., if the absolute value of any operand is equal to "1". If so, the output will be the other operand. However, **AxBy** will also modify the sign bit as the exclusive OR of sign bits from both operands. If there is no match for any type of trivial computation, the result will be the product of the original operands without any approximation.

V. Experimental Results

A. Experiment Setup

We write **AxBy** hardware modules in Verilog. We then synthesize them using Synopsys Design Compiler and place-and-route using Synopsys IC Compiler with TSMC 45nm technology. The baseline computation units are generated from FlopoCo [4] and synthesized and placed-and-routed by the same flow. We use Synopsys PrimeTime to evaluate and measure the power and area of the circuits under 1.0V. We select seven common applications that are computation-intensive, including four image processing applications: Roberts filter, Scharr filter, Sobel filter and Sharpen filter of the Leeds Butterfly datasets [26], and three CNN applications of the MNIST, EMNIST and CIFAR-10 datsets.

B. Hit Rate

1) Exact Pattern Matching: We measure the hit rate of trivial computations with exact matching for all applications as shown in Fig. 6 and can make several observations. First, for CNNs, the hit rates of multiplication are 35% to 40% for all three datasets. MNIST and EMNIST share similar addition hit rate of around 20% while CIFAR-10 has significantly higher hit rate of around 35% due to its different architecture. For image processing applications, the hit rates are significantly different across applications. Roberts filter has the highest hit rate among all the applications at around 55% while Sharpen filter hits only 15%. This is because these different operators have drastically different kernel values. For Robert filter, the kernel values are only "0"s and "1"s, which are naturally exact trivial computations.

2) Approximate Pattern Matching: To further enhance the computation bypass opportunity, we explore approximate trivial computations by using approximate pattern matching. However, not all the applications are suitable for approximate bypass. For image processing applications, using approximate bypass results in significant accuracy loss. Actually, with truncating only one bit (i.e., 31-bit matching), the accuracy loss reaches nearly 100% across all four image processing applications. This is because they are using integer data type in arithmetic operations. And one bit difference can induce an error of $1/256$ of the entire 8-bit RGB value range, making the errors significant. In contrast, for CNNs which use floating point data types, of which the error tolerance is quite high. For example, with 18 bits truncated (14-bit matching), there is only around 1% accuracy loss. This is because floating point data follows the IEEE-754 format. Error introduced in truncating mantissa bits are infinitesimal considering the broad range of floating point numbers.

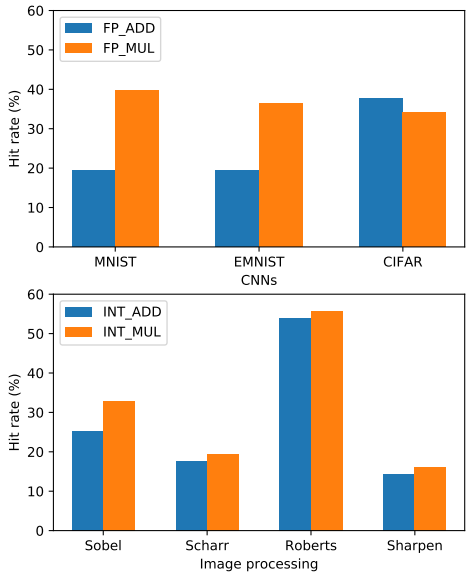


Fig. 6. Hit rate of trivial computations across different applications under exact bypass.

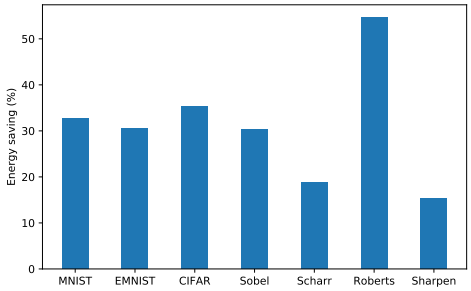


Fig. 7. Energy saving of **AxBy** across different applications under exact bypass.

It is clear that more aggressive pattern matching, i.e., fewer bits used for matching, will result a higher hit rate and energy savings but greater loss in accuracy. To investigate such tradeoffs, we choose four pattern matching settings: 8-bit, 10-bit, 12-bit and 14-bit. Fig. 8 illustrates the hit rate and accuracy loss under these four settings for all three datasets, based on which we can make several observations. First, a lower bit-matching mode indicates a higher hit rate. This applies to both floating point ADD and MUL. For example, the hit rate increases from around 40% (14-bit) to around 60% (8-bit) for MUL for all three datasets. This is because a lower bit-matching mode actually relaxes the matching standard and can enhance the bypass opportunities of trivial computation.

Second, as the hit rate increases, the prediction accuracy, however, decreases. This is because the lower-bit matching

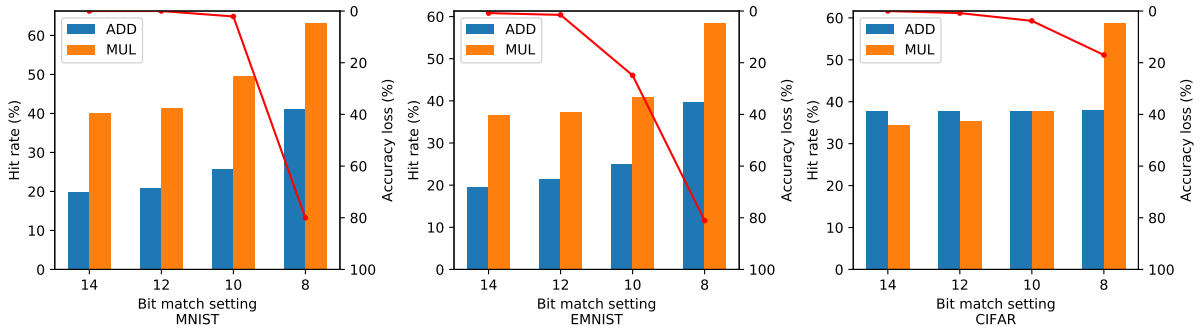


Fig. 8. Hit rate of trivial computations and accuracy loss across different applications under approximate bypass.

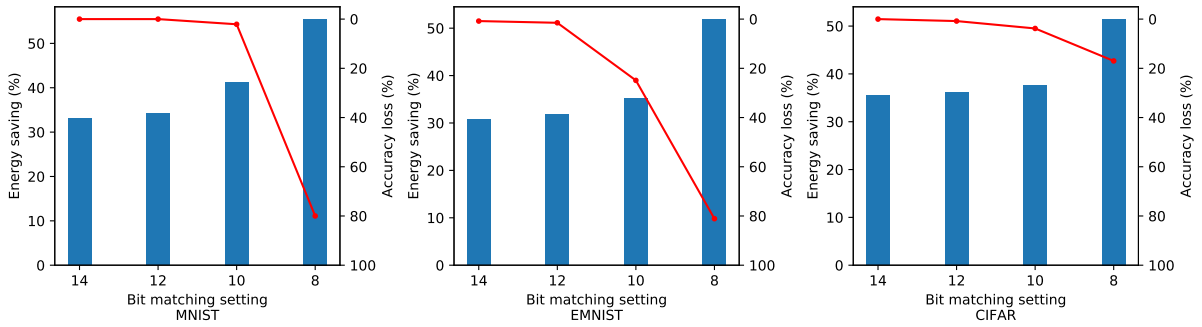


Fig. 9. Energy saving of **AxBy** under approximate bypass.

setting relaxes the matching standard and introduces more approximation errors when computing. Fortunately, by controlling the matching mode, we can actually control the accuracy loss because CNNs have a certain tolerance to errors. For example, the accuracy loss is negligible from 14-bit matching to 12-bit matching for all three datasets. Additionally, the accuracy loss dramatically increases when using 8-bit matching. This is because the 8^{th} bit position in a floating point number belongs to the exponential field. This will drastically increases the approximation error.

C. Energy Saving

1) Energy of Bypass Circuits: Table IV presents the enery consumption of each arithmetic operation and **AxBy** circuits. Note that we use the average energy consumption of **AxBy** circuits under four pattern matching settings: 8-bit, 10-bit, 12-bit and 14-bit. Actually, their energy consumption varies little (<3%). Based on Table IV, we can find that **AxBy**'s energy consumption ranges from 0.25% to 1% of energy of integer or floating point units. We also measure the area of bypass circuits. Compared to computation units, bypass circuits consume less than <3% area overhead on average across all types. Since computation units are universally deployed cross different platforms such as CPU, GPU, and ASIC, our approach is also architecture-independent.

We compute the energy saving of all the applications with **AxBy** compared to the baseline design with processing of normal computations unit based on Eq. 1, where r stands for the corresponding hit rate of one operation while E_{CU} and E_{AxBy} stands for the energy consumption of computation units and **AxBy**, respectively.

TABLE IV
PER OPERATION ENERGY CONSUMPTION OF COMPUTATION UNIT AND **AxBy**.

computation		energy (FJ)
INT	MUL	1377
	ADD	598
FP	MUL	4100
	ADD	2145
AxBy_FP	MUL	6.291
	ADD	6.051
AxBy_INT	MUL	14.685
	ADD	6.051

$$E_{sav}(\%) = \frac{(1-r)E_{CU} + rE_{AxBy}}{E_{CU}} \times 100\% \qquad (1)$$

2) Energy Saving of Exact Bypass: Fig. 7 shows the total energy savings with **AxBy** on different applications under exact bypass mode, i.e., no accuracy loss. The average energy saving across all seven applications is 31.1%. All CNNs can achieve energy savings of over 30%. Image processing applications show a varying degree in energy savings because their hit rates vary, as shown in Fig. 6. Clearly, Roberts filter has the highest hit rate, hence it also achieves the highest energy saving of all image processing applications. Actually,

the energy savings profile is very similar to the hit rate profile by examining Fig. 7 and Fig. 6.

3) Energy Saving of Approximate Bypass: To further enhance energy savings, we use approximate bypass with some reduced number of bits for pattern matching, as shown in Fig. 9. Under the 14-bit matching setting, there is nearly no accuracy loss and the energy savings are similar to that of exact matching. Then, as we reduce the number of bits for matching, the energy savings keep increasing. For example, when we use 10-bit matching, the energy savings increase to 40% from 33% (14-bit) for MNIST, with less than 2% accuracy loss. We can observe similar cases in EMNIST and CIFAR dataset that with a less than 2% accuracy loss, we enable 4%-6% more energy savings generally. However, we cannot go beyond 8-bit matching because that will incite significant accuracy loss up to 80%.

D. Design Space Exploration

We notice that we can further increase the energy savings with less accuracy loss if we use different matching modes for ADD and MUL. We conduct a sensitivity analysis to find out which operation has a greater impact on the quality when they are approximated. To do this, we perform a controlled experiment. That is, we change the matching mode of ADD while keeping the matching model of MUL at 32-bit (i.e., exact bypass), and then vice versa, to observe the corresponding accuracy loss. Fig. 10 presents the results of sensitivity analysis. For the MNIST and EMNIST datasets, ADD has higher sensitivity than MUL as its approximation leads to a higher accuracy loss. For the CIFAR dataset, MUL has higher sensitivity. This in fact emphasize the importance of designing reconfigurable and controllable hardware approximation to fit dataset-specific scenario.

We thus perform design space exploration to examine the effects of using different matching modes for ADD and MUL. As shown in Table V, after design space exploration, We can observe that with only accuracy loss at around 3%, **AxBy** can achieve 36.3% to 44.2% energy saving across three CNN applications. This is 3% - 5% higher than uniform bypass.

VI. Related Works

Approximate Computation Reuse Due to strong value locality and similarity presented in data-intensive applications, computation reuse has been exploited to improve efficiency [15], [14], [23], [12] by reusing the previously computed results to avoid redundant executions. For example, a complicated layer-level reuse mechanism was used in [23] to utilize the input similarity in a streaming applications such as speech. Weight similarity was leveraged to enabled a dot product factorization and activation group reuse [12]. Many computation reuse mechanisms work on the level of individual arithmetic operations. Frequent input patterns are profiled offline and then prestored in a look-up table, which was implemented in Bloom filters [15], STT-RAM based TCAM [14], and FeFET-based TCAM [28]. Such computation reuse was further enhanced by enabling approximate pattern matching under limited data precision. However, the most significant disadvantage of this approach is that it requires an offline profiling for each application and dataset to look for the most frequent input patterns. This is notoriously time-consuming and cost-prohibitive.

Pruning and Quantization Early success of hardware accelerators leverages pruning-based techniques to shrink the original dense network size by removing the neuron connections with weights below a certain threshold. This leads to a sparse network structure [9], [10]. Quantization is further used to compress the pruned network by using fewer bits for weights or activations. It typically converts the floating point parameters to fixed point parameters to reduce the computation workload of CNNs [20], [8]. While effective, these methods typically require *retraining* or fine-tuning to regain accuracy. This process lacks flexibility and can be time-consuming. Even with retraining, these methods can still incur accuracy loss.

Ineffectual Computation A research direction similar to **AxBy** is to exploit the *ineffectual multiplications* in CNNs [1], [18], [17]. Ineffectual multiplications are defined as multiplication with one of its input operands being zero (zero-aware multiplications). These multiplications can be skipped because once any of the two operands is zero, their product can be instantly determined as zero without actual operations. Therefore, by simply skipping these ineffectual multiplications, it is possible to reduce the computation workload thus enhance the energy efficiency on these applications. This set of approaches, however, is only limited to multiplications with an input operand at zero.

Main Difference Compared to these approaches, **AxBy** does not require offline profiling or retraining. Furthermore, **AxBy** enhances the scope of computation bypass by exploring various trivial computation classes and enabling approximate pattern matching. **AxBy** is also an architecture-independent approach due to its low energy/area overhead.

TABLE V
Recommended bit matching settings for **AxBy** in CNN applications

Application	ADD	MUL	Energy Saving (%)	Accuracy Loss (%)
MNIST	10	9	44.2	2.09
EMNIST	11	10	36.3	3.13
CIFAR	8	11	39.6	3.31

VII. Conclusion

In this work, we exploit the computation bypass opportunities in various data-intensive applications and enhance such opportunities by performing approximate pattern matching. We design **AxBy**, an approximate computation bypass approach with pattern matching under limited data precision. We design and implement bypass circuit architecture to physically implement the approximate pattern matching. By detecting trivial computation patterns, **AxBy** can bypass trivial computations to avoid the overhead caused by redundant executions on computation units. We enhance the opportunities of computation bypass by enabling pattern matching under

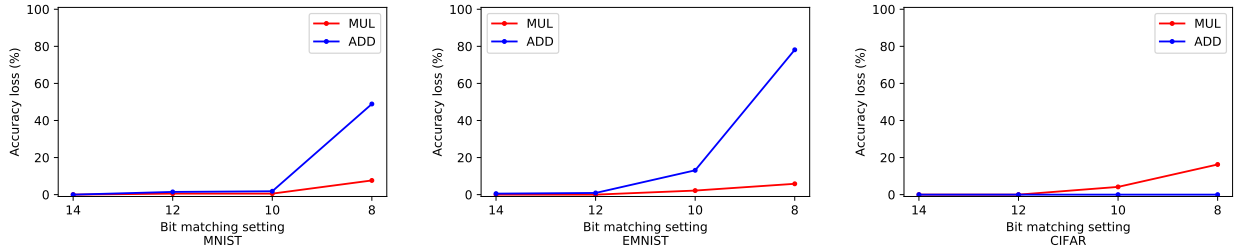


Fig. 10. Sensitivity analysis of CNN applications.

limited data precision. This reconfigurability is key to achieving a "controllable approximation" and a tunable quality-energy tradeoff. We evaluate the effectiveness of the proposed approach on four image processing applications and three CNN applications. Experimental results show that **AxBy** can enable 15% – 55% energy saving in image processing and 30% – 35% in CNN applications without any accuracy loss. For CNN applications, we can further achieve 36% – 44% energy savings with only negligible accuracy loss.

REFERENCES

[1] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 1–13. IEEE Press, 2016.

[2] Chia-Yu Chen, Jungwook Choi, Kailash Gopalakrishnan, Viji Srinivasan, and Swagath Venkataramani. Exploiting approximate computing for deep learning acceleration. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 821–826. IEEE, 2018.

[3] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 367–379. IEEE, 2016.

[4] Florent De Dinechin and Bogdan Pasca. Designing custom arithmetic data paths with flopoco. *IEEE Design & Test of Computers*, 28(4):18–27, 2011.

[5] Zidong Du, Avinash Lingamneni, Yunji Chen, Krishna V Palem, Olivier Temam, and Chengyong Wu. Leveraging the error resilience of neural networks for designing highly energy efficient accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(8):1223–1235, 2015.

[6] Walaa El-Harouni, Semeen Rehman, Bharath Srinivas Prabakaran, Akash Kumar, Rehan Hafiz, and Muhammad Shafique. Embracing approximate computing for energy-efficient motion estimation in high efficiency video coding. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1384–1389. IEEE, 2017.

[7] Amirali Ghofrani et al. Associative memristive memory for approximate computing in gpus. *IEEE JETCAS*, 2016.

[8] Philipp Gysel. Ristretto: Hardware-oriented approximation of convolutional neural networks. *arXiv preprint arXiv:1605.06402*, 2016.

[9] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 243–254. IEEE, 2016.

[10] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Kartik Hegde, Jiyong Yu, Rohit Agrawal, Mengjia Yan, Michael Pellauer, and Christopher W Fletcher. Ucnn: Exploiting computational reuse in deep neural networks via weight repetition. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pages 674–687. IEEE Press, 2018.

[13] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[14] Mohsen Imani et al. Efficient neural network acceleration on gpgpu using content addressable memory. In *DATE*. IEEE/ACM, 2017.

[15] Xun Jiao, Vahideh Akhlaghi, Yu Jiang, and Rajesh K Gupta. Energy-efficient neural networks using approximate computation reuse. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1223–1228. IEEE, 2018.

[16] Xun Jiao, Mulong Luo, Jeng-Hau Lin, and Rajesh K Gupta. An assessment of vulnerability of hardware neural networks to dynamic voltage and temperature variations. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 945–950. IEEE, 2017.

[17] Dongyoung Kim, Junwhan Ahn, and Sungjoo Yoo. A novel zero weight/activation-aware hardware architecture of convolutional neural network. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1462–1467. IEEE, 2017.

[18] Dongyoung Kim, Junwhan Ahn, and Sungjoo Yoo. Zena: Zero-aware neural network accelerator. *IEEE Design & Test*, 35(1):39–46, 2018.

[19] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, page 20, 2015.

[20] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858, 2016.

[21] Vojtech Mrazek, Syed Shakib Sarwar, Lukas Sekanina, Zdenek Vasicek, and Kaushik Roy. Design of power-efficient approximate multipliers for approximate artificial neural networks. In *2016 International Conference On Computer Aided Design (ICCAD)*, page 7, 2016.

[22] Padmanabhan Pillai and Kang G Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 89–102. ACM, 2001.

[23] Marc Riera, Jose-Maria Arnau, and Antonio González. Computation reuse in dnns by exploiting input similarity. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pages 57–68. IEEE Press, 2018.

[24] Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, and Joshua Zhexue Huang. Big data analytics on apache spark. *International Journal of Data Science and Analytics*, 1(3-4):145–164, 2016.

[25] Swagath Venkataramani, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. Axnn: energy-efficient neuromorphic systems using approximate computing. In *International symposium on Low power electronics and design*, pages 27–32. ACM, 2014.

[26] Josiah Wang, Katja Markert, and Mark Everingham. Learning models for object recognition from natural language descriptions. In *Proceedings of the British Machine Vision Conference*, 2009.

[27] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1492–1500, 2017.

[28] Xunzhao Yin, Michael Niemier, and X Sharon Hu. Design and benchmarking of ferroelectric fet based tcam. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1444–1449. IEEE, 2017.