TruLook: A Framework for Configurable GPU Approximation

Ricardo Garcia*, Fatemeh Asgarinejad*, Behnam Khaleghi*, Tajana Rosing*, and Mohsen Imani[†]⋆
*University of California San Diego, [†]University of California Irvine
⋆Email: m.imani@uci.edu

***Abstract*—In this paper, we propose TruLook, a framework that employs approximate computing techniques for GPU acceleration through computation reuse as well as approximate arithmetic operations to eliminate redundant and unnecessary exact computations. To enable computational reuse, GPU is enhanced with small lookup tables that are placed close to the stream cores that return already computed values for exact and potential inexact matches. Inexact matching is subject to a threshold controlled by the number of mantissa bits involved in the search. Approximate arithmetic is provided by a configurable approximate multiplier that dynamically detects and approximates operations which are not significantly affected by approximation. TruLook guarantees the accuracy bound required for an application by configuring the hardware at runtime. We have evaluated TruLook efficiency on a wide range of multimedia and deep learning applications. Our evaluation shows that with 0% and less than 1% quality loss budget, TruLook yields on average 2.1× and 5.6× energy-delay product improvement over four popular networks on the ImageNet dataset.**

I. Introduction

The past decades have witnessed the exponential growth of data production where over 90% of the available data today, has been produced in the last two years [1], [2], [3]. Evidently, the so-called data explosion outpaces the performance boost offered by technology scaling, hence making traditional processors incapable of keeping up with this large amount of produced data [4], [5]. One key aspect of many applications is their tolerance of error, meaning they can accept a certain level of error within their intermediate computations or even final results [6], [7], [8], [9]. For instance, in several computer vision applications, the exact output is not required as humans cannot perceive small visual distortion. Web searching service is another example where producing related results is crucial, but the order of the results is not always the primary concern. Machine learning algorithms, e.g., Deep Neural Networks (DNNs), have also shown considerable tolerance to error due to the pooling layers that filter out a majority of computation, as well as the stochastic nature of gradient [10], [11], [12], [13]. In all these examples, approximate computing is a viable solution to take advantage of error tolerance to exchange accuracy for energy-saving and/or performance boosting. Particularly, these applications are continuously growing in size, demanding more power-hungry GPU acceleration [14], [15].

The inherent temporal locality in machine learning applications has shown promising energy reduction attributes through computation reuse [16], [17], [18]. This can be enabled by associative memory that stores highly frequent arithmetic operations and avoids repeated computation. Previous works exploited computational reuse to enable GPU approximation, where associative memory stores the highly frequent patterns beside each GPU floating-point unit. However, it is unlikely to find an exact or even approximate match over the entire exponents and mantissa bits of pre-stored operands [16]. This significantly reduces the associate memory hit rate and eliminates the advantage of approximation.

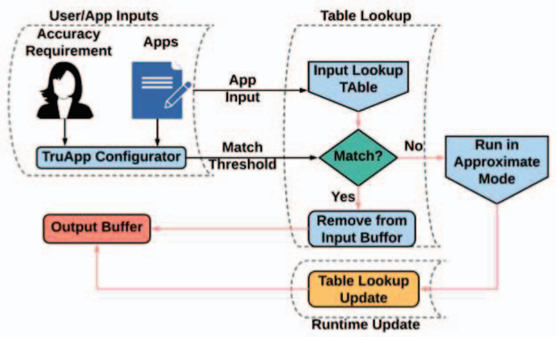


Fig. 1. TruLook framework enabling GPU approximation.

An orthogonal approach for realizing approximation is to employ approximate arithmetic units [19], [20]. These units identify the input operands with minimal impact on the computation accuracy when approximated. The configurable floating-point unit executes the operation in the approximate mode. The main drawback of these approaches is having low control over the quality loss, especially when the overall quality budget is tight (e.g., $<2\%$ error).

In this paper, we present TruLook, a configurable approximate computing framework that targets accelerating GPU applications. TruLook enables each GPU core to leverage data locality, whilst it approximates benign input operands, as well. For data locality, TruLook utilizes a small lookup Table located near the stream core for enjoying computational reuse. Rather than relying on the approximate match on the entire input operands, TruLook only checks for the similarity of the mantissa bits, while computes the sign and exponent bits of the result in exact mode. This method not only improves the hit rate of the lookup Table by reducing the design space (for a given lookup Table size), but also enhances computation accuracy by eliminating inexact matches on the critical exponent and sign bits. For approximate arithmetic, we propose a configurable approximate multiplier to detect and approximate operands that are less susceptible to error. TruLook ensures high accuracy through runtime hardware configuration based on the quality determined by the user.

II. Proposed TruLook

A. Overview

We design a novel framework that enables users to determine the level of accuracy in exchange for higher efficiency. Figure 1 illustrates the overview of TruLook's framework. Our framework receives specifications from the user to determine the maximum error allowed. The application and a queue of the application's inputs are imported into the data analysis. During the data analysis, a variety of computation takes place, including multiplication and Lookup Table M bits matching. When an input is fed into the design, it is first checked whether

978-3-9819263-5-4/DATE21/©2021 EDAA

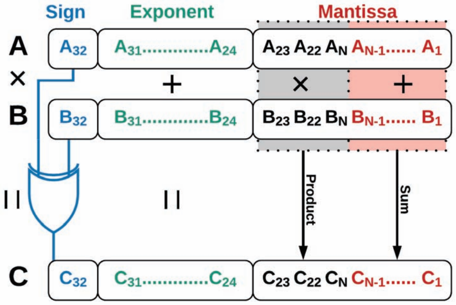

Sign
Exponent
Mantissa
A
×
B
C
A_{32}
A_{31}.............A_{24}
A_{23} A_{22} A_N A_{N-1}...... A_1
+
×
+
B_{32}
B_{31}.............B_{24}
B_{23} B_{22} B_N B_{N-1}...... B_1
||
||
Product
Sum
C_{32}
C_{31}.............C_{24}
C_{23} C_{22} C_N C_{N-1}...... C_1

Fig. 2. TruLook approximate multiplication.

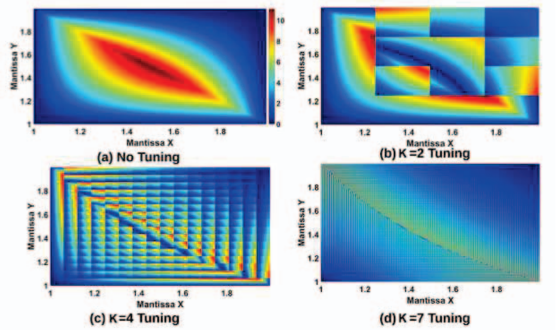
a close approximate match already exists in the lookup table, so the value can be retrieved from the memory and bypassed to the output. Otherwise, the inputs are passed to an approximate multiplier, the result is forwarded to the output, and the lookup Table is updated with the inputs and produced the result. As explained earlier, our approach only checks the mantissa bits similarity and calculates the exponents and sign by itself to increases the hit rate and avoid extreme accuracy loss.

B. Approximate Arithmetic

We use IEEE 754 32-bit floating-point notation, which is represented as a 32-bit binary number $(A_{32}, \ldots, A_1)$ with three different parts; a sign bit, an exponent part, and a fractional value. The first bit in the floating-point notation A_{32} represents the sign bit. The next eight bits, $A_{31}, \ldots, A_{24}$, represent the exponent of the binary numbers ranging from -126 to 127. The following 23 bits, $A_{23}, \ldots, A_1$, represent the fractional part or mantissa and have a value between 1 and 2.

To improve the efficiency of approximate multiplication, several methods have introduced the idea of removing or replacing mantissa multiplication. The state-of-the-art approximate multiplier, RMAC [20], replaces (approximates) the mantissa multiplication with the mantissa addition. Although this method affords a very low initial error rate, the accuracy cannot be well-tuned. RMAC is only capable of reducing the maximum error to approximately 6% error, which is typically accomplished through using the result of the approximation by looking at the count of a consecutive number of '1's and '0's in the result. On the other hand, our approach focuses on using a combination of a small variable multiplier with an adder. The design is shown in Figure 2, where $23 - N$ bits are being multiplied in the exact mode, and the remaining N bits are being summed using an adder. Simply replacing the multiplications of mantissa with an adder (i.e., $N = 23$) results in a maximum error of 11.11%. The value of N dictates the partial multiplier's accuracy level, where a lower N value results in higher accuracy and vise versa. Our focus is on reducing the multiplier size as much as a high accuracy can still be maintained without using a full multiplier. As it pertains to the exponent and sign bits, they are not partitioned. The sign bits of both inputs are XORed, and the exponents are added together.

As multiplication and addition parts are independent, they are done in parallel. The main reason for the dual multiplier-adder implementation is based on the behavior of the error rate. A stand-alone multiplier omitting the least significant mantissa bits has a high error, which can be fairly compensated, letting the adder care of remaining values. Figure 3 shows the heat map of the error rate for the different levels of accuracy that design can achieve for all possible partial multiplications. The trend we see is like the $K = 23 - N$ (i.e., the size of the multiplication part) increases, the probability of large errors diminishes. It remarkably enhances the accuracy level that our design is able to hit.



(a) No Tuning

(b) K =2 Tuning

(c) K=4 Tuning

(d) K =7 Tuning

Fig. 3. TruLook error distribution for different tuning parameter K.

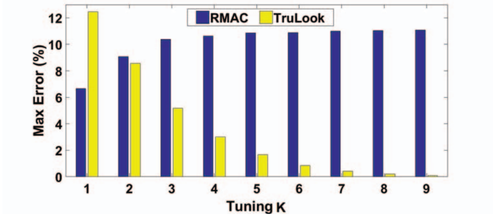


Fig. 4. TruLook and RMAC [20] maximum error for different parameter K

Figure 4 compares TruLook and RMAC maximum error for different values of K. For our design, $K = 23 - N$ shows the size of the multiplier, in which N is the size of the accompanying adder. For RMAC, K shows the approximation level (the smaller K, the better accuracy). As the figure reveals, RMAC maximum error rate never goes better than 6%, compared to TruLook, which has a maximum error of 0.09% at $K = 9$ while further improved by increasing K. As it pertains to a maximum overall error in worst-case configuration, RMAC has a slightly lower error with 11.11%. In contrast, our design has a maximum error of 12.46% in its worst configuration. Another key difference between the two designs is how they benefit as the degree of approximation reduces. For the RMAC, the error difference between $K = 9$ and $K = 3$ is 0.71%; thus, RMAC does not scale well by tuning the approximation level. In contrast, our design has a higher error difference between K values than RMAC. This is a crucial characteristic considering application requirement, which makes TruLook to make worthwhile approximation-gain trade-offs at different application requirements.

C. Computational Reuse

The idea of computational reuse is to exploit a lookup Table to store frequently used input/output operands and eliminate redundant computations. The input values are matched exactly or approximately with the lookup Table values. Typically, when utilizing computational reuse, the whole 32-bits or 64-bits floating-point values are stored in the lookup table, which results in a very low hit rate even in the approximate mode. Using the static lookup Table is another reason for the low hit rate, further degrading the efficiency. We propose

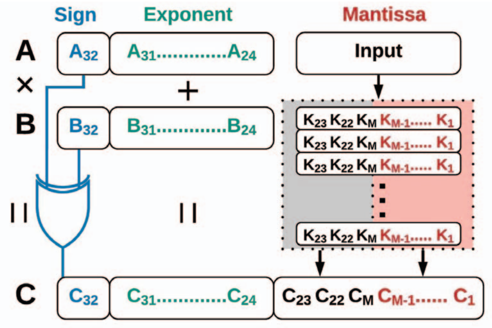

Sign
Exponent
Mantissa
A
A_{32}
A_{31}............A_{24}
Input
×
+
B
B_{32}
B_{31}............B_{24}
K_{23} K_{22} K_M K_{M-1}..... K_1
||
||
C
C_{32}
C_{31}............C_{24}
C_{23} C_{22} C_M C_{M-1}..... C_1

Fig. 5. TruLook computational reuse scheme.

two techniques to improve the lookup Table hit rate: variable mantissa matching and adaptive lookup table.

Mantissa Matching: TruLook increases the lookup Table hit rate by only matching the mantissa of the input operands by M number of bits. Meanwhile, TruLook reconstructs the exponent of output by simply adding the exponents of the input operands. Figure 5 shows the overview of TruLook computational reuse. TruLook, first, takes the input tunable variable M to establish the maximum difference between the stored values and the incoming input. Once the maximum allowable difference is determined, the new inputs mantissa are compared to the stored lookup Table values. If the input lies within a specified difference range of the lookup Table values, the corresponding stored key in the lookup Table is fetched and directed to the output.

Adaptive Lookup Table: Another challenge in reusing computation is selecting the appropriate Table values. GPUs are running applications with different types of workloads, which themselves might also change during time. For instance, GPUs processing images of different seasons may have different frequent input patterns due to the different background colors of seasons. As a result, updating the lookup Table values at the runtime is inevitable. Previous work has proposed using reinforcement learning to find the highly frequent patterns at the runtime [21]. The learning method itself, however, increases the computation cost. In contrast, we employ a straightforward Least Recently Used (LRU) policy to dynamically update the lookup Table values in the course of runtime. Our LRU policy assigns a 4-bit counter to each of the lookup Table entries to record their usage frequency. The lookup Table entries with minimum reuse are the candidate for replacement with a new pattern. The LRU cost can be amortized in different ways, e.g., calibrating the counter's size or allocating a single counter for a bunch of entries (particularly for entries with close values).

III. Experimental Results

A. *Experimental Setup*

To implement TruLook, we use a modified version of Multi2Sim [22], which is a CPU-GPU simulator. We modified the kernel code in order to implement the proposed design for runtime simulations. To demonstrate the versatility of TruLook, we employed two different GPU architectures: the AMD Southern Island Radeon HD 7970 and the Nvidia Kepler GeForce GTX Titan devices, and integrated TruLook within the floating-point units. We characterized the multiplier and adder by synthesizing them using 45nm NanGate Open Cell Library [23] in Synopsys Design Compiler. We evaluated TruLook on four multimedia benchmarks from m2s-bench-cudasdk-6.5 [22], including BoxSharp, Sobel, Prewitt, and Robert. For these benchmarks, we use Peak Signal to Noise Ratio (PSNR) as the accuracy metric.

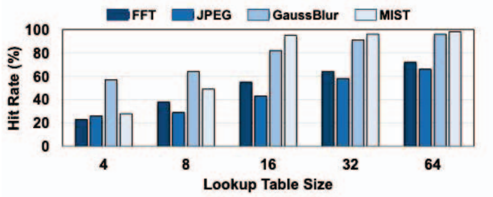


Fig. 6. Impact of lookup Table size on the computation reuse hit rate ($M = 4$ is used for all the evaluations).

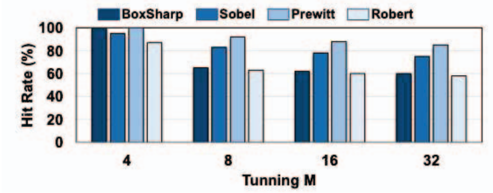


Fig. 7. Impact of tuning parameter M on the hit rate of different applications in a lookup Table with 52 entries.

B. TruLook *Computational Reuse*

Figure 6 shows the hit rate using different lookup Table sizes for a fixed tuning parameter of $M = 4$. Remember that M denotes the number of mantissa bits used for lookup. The high hit rate value indicates a promising potential of using computation reuse for approximation. As expected, the hit rate is correlated with the size of the lookup table. A large lookup Table increases the number of approximated computations due to the wider range of pre-stored inputs. The increase in hit rate is not linear, and for most of the evaluated applications, it saturates for tables larger than 32 entries. In addition, larger tables require higher energy for search operation. That being said, we found tables with 16 entries as optimal Table size.

Figure 7 shows the impact of the tuning parameter M on the lookup Table hit rate for a Table with a fixed size of 52 for 64-bit floating-point representation. As expected, increasing the M value results in a lower hit rate as it degrades the matching probability. A lower M value increases the number of least significant mantissa bits that can be overlooked during the similarity search. On the other hand, a lower value of M incurs a higher degree of approximation and potentially leads to larger accuracy degradation. Table I and Table II summarize the impact of tuning parameter M on the quality and the efficiency of the computation. Accordingly, a low value of M provides low computation accuracy, while the computation efficiency is maximized. To provide an acceptable quality of computation, based on Table I, TruLook requires to use $M \geq 16$. In such a configuration, TruLook improves the energy efficiency by 5.5×, and performance by 1.3× as compared to the baseline GPU.

C. TruLook *Approximate Arithmetic*

Figure 8 shows the impact of TruLook tuning parameter K (size of the multiplier) on the hit rate of the approximate arithmetic at $M = 16$. Using a larger K tuning parameter decreases the degree of approximation as it increases the

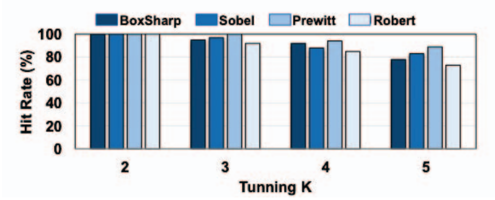


Fig. 8. Impact of tuning variable K on the hit rate of different applications for a fixed $M = 16$.

TABLE I
QUALITY OF COMPUTATION (PSNR) FOR DIFFERENT TUNING PARAMETERS (NUMBERS ARE IN db SCALE).

	Tuning M				**Tuning K**				
Application	**4**	**8**	**16**	**32**	**2**	**3**	**4**	**5**	**8**
BoxSharp	28.8	38.6	48.6	62.4	13.3	21.8	27.1	32.9	44.9
Sobel	28.41	35.3	42.9	59.4	16.5	19.6	26.8	32.1	39.7
Prewitt	30.50	37.6	51.3	68.0	18.9	21.6	28.2	34.1	47.5
Robert	35.8	40.1	63.2	73.6	24.6	27.2	34.5	38.9	51.1

TABLE II
ENERGY AND PERFORMANCE IMPROVEMENT OF TruLook FOR DIFFERENT TUNING PARAMETERS.

	Tuning M				**Tuning N**				
Application	**4**	**8**	**16**	**32**	**2**	**3**	**4**	**5**	**8**
Norm.Energy	0.09	0.18	0.46	0.54	0.55	0.27	0.39	0.72	0.83
Norm. Execution	0.69	0.77	0.89	0.92	0.74	0.79	0.86	0.92	0.98

multiplier size. It, however, results in a higher computation accuracy with a limited impact on computation reuse. The impact of tuning parameter K can also be observed in Table I. A larger value of K results in higher energy consumption and lower performance, though it improves the accuracy until it is satisfied at $K = 8$. In terms of efficiency, TruLook using a smaller K value provided higher computation efficiency.

D. TruLook & Deep Learning

Table III shows TruLook energy-delay product (EDP) improvement on popular Deep Neural Networks (DNNs). Using ImageNet dataset [24], we have evaluated TruLook efficiency on four popular networks including, AlexNet [24], VGGNet [25], GoogleNet [26], and SqueezeNet [27]. For a given accuracy loss budget, we have tuned the parameters to yield the maximum EDP improvement. The result shows that TruLook provides, on average, 2.1× (5.7×) EDP improvement while ensuring 0% (1%) quality loss. This efficiency comes from redundancy in the DNN computation, which provides a superb potential for computational reuse. Particularly, the efficiency of TruLook is more pronounced on networks with a larger number of convolution layers, as these networks experience a larger amount of redundant computation. Another reason for the improvement is due to the fact that DNN involves a large number of multiplications with low or no impact on network accuracy, such as the pooling layers that discard a major portion of computation results. In addition, DNNs are stochastic in their nature, so a slight addition of noise can be perceived as a perturbation of the network parameters, for which DNNs are resilient to some extent.

IV. CONCLUSION

In this paper, we propose TruLook, a hybrid approximate GPU architecture. To enable computational reuse, the GPU is augmented with small lookup tables integrated within the stream cores. To realize approximate arithmetic, TruLook proposes a configurable approximate multiplier that dynamically detects and approximates operations that are less affected by approximation. Our framework ensures the accuracy required by application by configuring the hardware at runtime.

TABLE III
TruLook EDP IMPROVEMENT FOR DIFFERENT DNN NETWORKS AND ERROR BUDGETS.

Quality Loss	**0%**	**0.1%**	**0.5%**	**1%**	**2%**
AlexNet	1.6×	2.3×	2.9×	3.7×	6.2×
VGGNet	1.9×	2.8×	3.5×	4.7×	7.4×
GoogleNet	2.7×	4.6×	5.8×	8.5×	10.3×
SqueezeNet	2.0×	3.1×	4.0×	5.9×	8.1×

TABLE IV
COMPARING THE HIT RATE AND EDP IMPROVEMENT OF TruLook WITH PREVIOUS WORK FOR ONE MILLION RANDOMLY GENERATED NUMBERS.

Error Rate		**1%**	**2%**	**4%**	**6%**	**8%**	**10%**
CFPU [28]	Hit Rate	3.4%	7.5%	14.7%	19.4%	31.1%	37.2%
	EDP Improve	1.03×	1.07×	1.16×	1.22×	1.41×	1.54×
RMAC [20]	Hit Rate	0%	0%	0%	93.7%	99.9%	100%
	EDP Improve	0×	0×	0×	6.80×	11.07×	11.31×
TruLook	Hit Rate	58.36%	94.06%	100%	100%	100%	100%
	EDP Improve	1.8×	2.9×	3.7×	10.3×	15.8×	18.0×

ACKNOWLEDGMENT

This work was partially supported by Semiconductor Research Corporation (SRC) Task No. 2988.001.

REFERENCES

[1] C. Dobre and F. Xhafa, "Intelligent services for big data science," *Future Generation Computer Systems*, vol. 37, pp. 267–281, 2014.
[2] S. Sen *et al.*, "Approximate computing for long short term memory (lstm) neural networks," *IEEE TCAD*, vol. 37, no. 11, pp. 2266–2276, 2018.
[3] M. Imani *et al.*, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *ISCA*, pp. 802–815, IEEE, 2019.
[4] V. C. Storey *et al.*, "Big data technologies and management: What conceptual modeling can do," *Data & Knowledge Engineering*, vol. 108, pp. 50–67, 2017.
[5] M. Imani *et al.*, "Dual: Acceleration of clustering algorithms using digital-based processing in-memory," in *IEEE/ACM MICRO*, pp. 356–371, IEEE, 2020.
[6] S. Xu *et al.*, "Exposing approximate computing optimizations at different levels: From behavioral to gate-level," *IEEE TVLSI*, vol. 25, no. 11, pp. 3077–3088, 2017.
[7] C. Chen *et al.*, "Optimally approximated and unbiased floating-point multiplier with runtime configurability," in *IEEE/ACM ICCAD*, pp. 1–9, IEEE, 2020.
[8] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proceedings of the 52nd Annual Design Automation Conference*, p. 120, ACM, 2015.
[9] M. Imani *et al.*, "Ultra-efficient processing in-memory for data intensive applications," in *DAC*, p. 6, ACM, 2017.
[10] C.-Y. Chen *et al.*, "Exploiting approximate computing for deep learning acceleration," in *DATE*, pp. 821–826, IEEE, 2018.
[11] Y. Fan, X. Wu, J. Dong, and Z. Qi, "Axdnn: towards the cross-layer design of approximate dnns," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pp. 317–322, ACM, 2019.
[12] J. Choi and S. Venkataramani, "Approximate computing techniques for deep neural networks," in *Approximate Circuits*, pp. 307–329, Springer, 2019.
[13] S. Gupta *et al.*, "Scrimp: A general stochastic computing architecture using reram in-memory processing," in *DATE*, pp. 1598–1601, IEEE, 2020.
[14] H. Zhang *et al.*, "Poseidon: An efficient communication architecture for distributed deep learning on gpu clusters," in *SENIX ATC*, pp. 181–193, 2017.
[15] T. Chen *et al.*, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.
[16] M. Imani *et al.*, "Resistive configurable associative memory for approximate computing," in *DATE*, pp. 1327–1332, IEEE, 2016.
[17] D. Peroni *et al.*, "Arga: Approximate reuse for gpgpu acceleration," in *DAC*, p. 8, ACM, 2019.
[18] M. Imani *et al.*, "Approximate computing using multiple-access single-charge associative memory," *IEEE TETC*, 2016.
[19] M. Imani *et al.*, "Cfpu: Configurable floating point multiplier for energy-efficient computing," in *IEEE/ACM DAC*, pp. 1–6, IEEE, 2017.
[20] M. Imani *et al.*, "Rmac: Runtime configurable floating point multiplier for approximate computing," in *ISLPED*, p. 12, ACM, 2018.
[21] M. Imani *et al.*, "Acam: Approximate computing based on adaptive associative memory with online learning," in *IEEE/ACM ISLPED*, pp. 162–167, 2016.
[22] X. Gong, R. Ubal, and D. Kaeli, "Multi2sim kepler: A detailed architectural gpu simulator," in *ISPASS*, pp. 269–278, April 2017.
[23] "Nangate open cell library."
[24] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *NIPS*, pp. 1097–1105, 2012.
[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.
[27] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.
[28] M. Imani *et al.*, "Cfpu: Configurable floating point multiplier for energy-efficient computing," in *DAC*, pp. 1–6, IEEE, 2017.