# CADE: Configurable Approximate Divider for Energy Efficiency

Mohsen Imani, Ricardo Garcia, Andrew Huang, and Tajana Rosing
Computer Science and Engineering Department, UC San Diego, La Jolla, CA 92093, USA
{moimani, rag023, anh162, tajana}@ucsd.edu

***Abstract*—Approximate computing is a promising solution to design faster and more energy efficient systems, which provides an adequate quality for a variety of functions. Division, in particular, floating point division, is one of the most important operations in multimedia applications, which has been implemented less in hardware due to its significant cost and complexity. In this paper, we proposed CADE, a Configurable Approximate Divider which performs floating point division operation with a runtime controllable accuracy. The approximation of the CADE is accomplished by removing the costly division operation and replacing it with a subtraction of the input operands mantissa. To increase the level of accuracy, CADE analyses the first $N$ bits (called tuning bits) of both input operands mantissa to estimate the division error. If CADE determines that the first approximation is unacceptable, a pre-computed value is retrieved from memory and subtracted from the first approximation mantissa. At runtime, CADE can provide a higher accuracy by increasing the number of tuning bits. The proposed CADE was integrated on the AMD GPU architecture. Our evaluation shows that CADE is at least 4.1× more energy efficient, 1.5× faster, and 1.7× higher area efficient as compared to state-of-the-art approximate dividers while providing 25% lower error rate. In addition, CADE gives a new knob to GPU in order to configure the level of approximation at runtime depending on the application/user accuracy requirement.**

***Index Terms*—Approximate computing, Energy-efficiency, GPGPU**

## I. Introduction

The number of smart devices has been increasing exponentially over the past decade to a point where they outnumbered human beings [1]. As the Internet of Things (IoT) is realized, devices will be ready to react to a person's every desire. With humans still highly dependent on their senses, IoT systems would need to be able to interact with humans through embedded devices [2]. As a result, a large amount of data would be gathered from the interactions between humans and devices, requiring fast and real-time data processing [3]–[5]. This poses a challenge as current embedded devices lack resources and battery life to process enormous amount of data [3].

Current sensory data algorithms tend to be, at their core, statistical, capable of functioning with approximate computations [6]–[12]. As approximate computing continues to gain traction for its low energy consumption, precision would be replaced with energy efficiency in embedded devices [6]. Recently, attention has been focused on designing approximate arithmetic units such as adder [13]–[15] and multipliers [9], [16]–[18]. However, division, in particular, floating point division, is one of the most important operations in multimedia applications. For example, in the OpenCV library [19], a large amount of image processing applications utilize division in their computation. In current architectures, the division tends to be the least utilized due to its high energy consumption and low speed [20].

Recent work has tried to accelerate division by enabling approximation [21]–[25]. Work in [21], [22] focused on the truncation of bits to a better approximate division. Although this approach reduces the size of the required division, (i) it still utilizes a costly divider, and (ii) it does not offer a way to change the level of accuracy at runtime. Work in [22] approximate the division functionality by enabling a truncated value to be multiplied by an approximated inverse value of the dividend. Work in [26] changed the standard division functionality by incorporating new or modifying logic blocks in order to stray away from the division. This improves the divider area and energy efficiency by reducing the number of processing bits or area. However, these approaches do not include tuning methods that can be used at runtime. Thus, limiting the generality of these approaches to Application-Specific Integrated Circuit (ASIC), where the applications that require a lower error rate can be predetermined prior to chip design.

In this paper, a runtime configurable floating point divider, called CADE, is proposed capable of high error runtime correction without alienating specific inputs and simple integration into general processors. CADE replaces the floating point division with a subtraction of the two input operands mantissa. The methodology behind the design arises from analyzing the division process which consists of shifting and subtraction process that occurs. The design ensures less than 12.5% error rate with runtime approximate configuration with high energy efficiency performance. CADE was integrated as a new floating point unit in AMD GPU architecture. Our evaluation shows that CADE is at least 4.1× more energy efficient, 1.5× faster, and 1.7× higher area efficient as compared to state-of-the-art approximate dividers while providing 25% lower error rate. In addition, CADE creates a knob for multi-level configuration of approximation during runtime for the GPU to better adjust to each application error tolerance.

## II. Proposed CADE

### A. IEEE-754 Floating Point

Implementing the IEEE-754 32 bit floating point notation that is characterized as a 32-bit number string $(X_{32},\ldots,X_1)$ consisting of three distinct elements: a sign bit, exponent bits, and fractional value bits. The initial bit in the floating point notation $(X_{32})$ indicates the sign bit. The following 8 bits indicate the exponent of a binary number $(X_{31},\ldots,X_{24})$, with a range of -126 to 127. The final 23 bits $(X_{23},\ldots,X_1)$ indicate the fractional element, also referred to as the mantissa, with a value range between 1 and 2. (Figure 1)

978-3-9819263-2-3/DATE19/©2019 EDAA

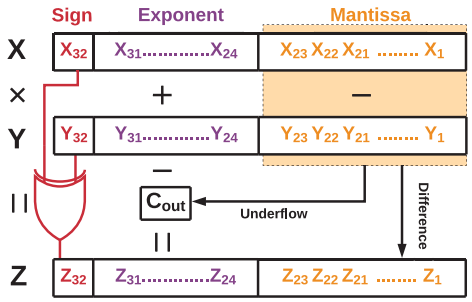



Fig. 1. Proposed division between X and Y floating point values.

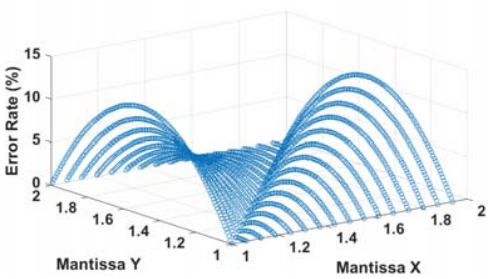



Fig. 2. The error distribution for $X$ divided by $Y$ as the mantissa values for $X$ and $Y$ increase from 1 to 2.

### B. Approximate Division

In this paper, a runtime configurable floating point divider, called CADE, is proposed that is capable of supporting approximate division. Figure 1 illustrates an overview of the CADE approximate model. The division process starts by XORing the sign bits of $X$ and $Y$ input operands, followed by the subtracting of both the exponent and mantissa elements of $X$ and $Y$. If mantissa $X$ is less than mantissa $Y$, subtraction will result in an underflow; to correct the underflow, an additional 1 is subtracted from the exponent. Thus, approximations are applicable to the fixed-point division between the mantissa because it can be achieved by subtraction and shifting of the operation between the partial products. Since in floating point representation, the shift is already applied by representing the mantissa with a value between 1-2, thus, this subtraction is a good approximation of the mantissa division.

Figure 2 shows the error distribution of the proposed divider design for $X$ and $Y$ input operands, with mantissa values ranging from 1 to 2. Assume the input $X$ mantissa is constant, thus in CADE, the division error reaches to its maximum 12.5% when mantissa $Y$ value increases to 1.5. From another hand, the error decreases to 0% as mantissa $Y$ reaches to value 2. The same trend occurs when $Y$ is constant and the $X$ mantissa increases to 2. Although the maximum error rate for the proposed divider without proper tuning is 12.5%, the error rate is highly dependent on its input operands. For example, if $X = 35$ and $Y = 10$ then in exact mode the division answer is 3.5. However, using CADE results in 3.69 approximation with an error rate of 5.36%. To further control the error, a tuning process that limits the maximum error rate that CADE accepts is required. This would allow all inputs to be run in approximate mode.

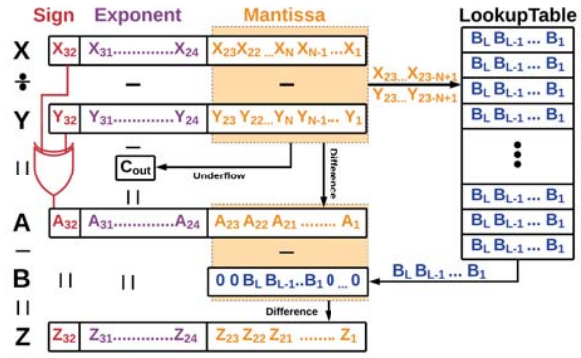
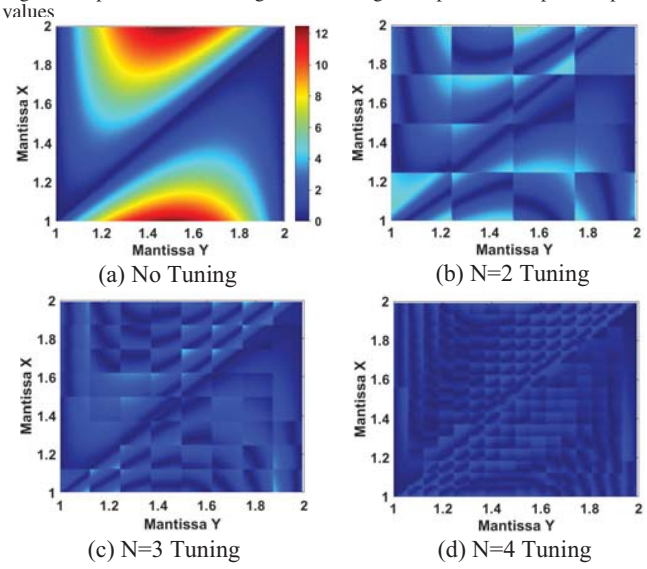



Fig. 3. Proposed CADE tuning method using lookup table with pre-computed values




(a) No Tuning (b) N=2 Tuning

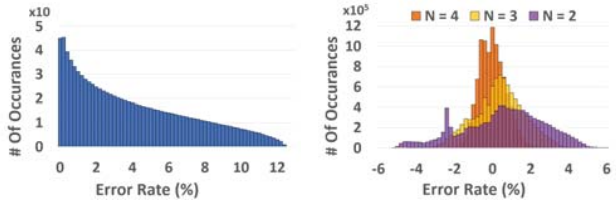
(c) N=3 Tuning (d) N=4 Tuning

Fig. 4. Heatmap error rate distribution for CADE with (a) no tuning, and (b-d) after tuning with $N$ =2, 3 and 4.

### C. Tuning Approximation

To achieve the desired accuracy, we design a tuning method which allows CADE to configure the maximum acceptable error rate. Figure 3 shows the overview of the CADE tuning approach. The tuning operation consists of a second stage approximation where the lookup table contains the offsets required to reduce the error of the CADE. For a given application, $X$ and $Y$ are the inputs, $A$ is the approximation with no tuning, $B$ is the selected offset, and $Z$ is the final approximation post tuning. Since the error is deterministic based on the inputs, specific input cases that produce high error can be detected and then corrected accordingly. As a result, CADE maintains the benefits of approximating the result without having to resort the exact computation for cases with a high error rate. The tuning design takes advantage of the mantissa representation where the most significant bits of each input mantissa have the most weight in determining the result of a computation in both exact and approximate modes. Thus, the $N$ most significant bits of each input mantissa are used to estimate the error that CADE produces.

The $N$ value is determined by the maximum error that an application is willing to accept. A higher $N$ value results in $2^{(N\times 2)}$ regions of resolution to tune for error, however, it




(a) (b)

Fig. 5. Distribution of error rate (a) before tuning (b) tuning in different configurations N# of tuning bits).

requires a larger lookup table to store the values utilized in the second stage of approximation. Each lookup table entry corresponds to a region with a maximum and minimum error range that is used to determine the optimal error correction offset. Figure 4a illustrates the error rate with no tuning through a heat map, where the error rate is highest when mantissa $X$ is both closer to 1 and 2 and mantissa $Y$ is in the range of 1.2 and 1.8. Through the detection of the values that produce the highest error rate, an adequate value of $N$ can be selected to decrease the error rate around the mantissa values of $X$ and $Y$ that produce a high error rate. Figure 4b-d show how the error rate around the mantissa values of $X$ and $Y$ decreases as the value of $N$ increased. Once a $N$ value is selected, the offset values are stored in a programmable table, rather than having them hardwired through combinational logic or a static table, this allows the design to be optimized for specific applications past the general case. For example, when dividing by power of 2's CADE requires no tuning; however, for an application that requires a lower error rate, the tuning method can be implemented to satisfy the application's error rate restrictions. The error distribution for random inputs with no tuning is illustrated in Figure 5*a* and with tuning in Figure 5*b* whereas the value of $N$ increases, the error distribution begins to center around 0.

## III. Results

### A. Experimental Setup

CADE was integrated into an AMD GPU architecture, Radeon HD 7970 device, by modifying Multi2sim, a cycle accurate CPU-GPU simulator [27]. We add CADE as a new floating point approximate divider. The FPUs are balanced for 6-stage using FloPoCo [28] and are synthesized by *Synopsys Design Compiler* in 45-nm ASIC flow and optimized for power consumption using *Synopsys Prime Time*. The energy consumption and execution time of the proposed CADE are measured using HSPICE circuit-level simulation in 45-nm technology. For the application level, the effectiveness of the proposed CADE was tested by running a wide range of applications three popular applications including image compression, image filtering, and Taylor series approximation.

### B. Design vs Other Dividers

Comparing the efficiency and accuracy of the proposed CADE with state-of-the-art dividers including the Low-Power Divider(LPDivider) [21], SEERAD [29], TruncApp [22], and AXDnr [23]. Our evaluation shows that CADE can achieve at least 4.1× energy efficiency improvement and 1.5× speedup as compared to prior work while providing 25% lower error rate.

TABLE I
Comparison of the proposed CADE with state-of-the-art approximate dividers.

| | Max Error | Energy (pJ) | Execution (ns) | Tunable | Runtime Tunable | Floating Support |
|---|---|---|---|---|---|---|
| **LPDivider** [21] | 36.0% | 3.17pJ | 5.70ns | ✗ | ✗ | ✗ |
| **SEERAD(1)** [29] | 37.5% | 0.75pJ | 0.76ns | ✓ | ✗ | ✗ |
| **Truncapp** [22] | 50.0% | 1.14pJ | 1.08ns | ✓ | ✗ | ✗ |
| **AXDur** [23] | 60.0% | 2.06pJ | 2.46ns | ✓ | ✗ | ✗ |
| **CADE** | 12.5% | 0.18pJ | 0.51ns | ✓ | ✓ | ✓ |

TABLE II
Average, Maximum, and Minimum error rate for CADE using different tuning bits.

| *N* | **0** | **1** | **2** | **3** | **4** |
|---|---|---|---|---|---|
| Average Error | 4.06% | 0.22% | 0.58% | 0.42% | 0.05% |
| Max Error | 12.5% | 9.01% | 6.03% | 4.72% | 3.03% |
| Lookup Size | NA | 4B | 16B | 64B | 256B |

Table I also compares different designs in terms configurability. Our evaluation shows that although most of the approaches can configure the level of accuracy, their configuration can only happen in offline. In contrast, to the best of our knowledge, CADE is the first floating point divider which can configure the level of accuracy at runtime depending on the application requirement. In addition, the lack of floating point support further reduces the generality of the prior work, for platforms such as CPU and GPU.

### C. CADE Exploration

**CADE & Tuning Bits:** In CADE, the number of tuning bits and lookup table bit-width provide a tradeoff between CADE accuracy and efficiency. The number of tuning bits, $N$, determines a granularity that we can detect different regions. Table II lists the average and maximum CADE error rate using a different number of tuning bits. Our result shows that both maximum and average CADE error decrease by increasing the number of tuning bits. However, as $N$ increases, CADE requires more memory to store the pre-computed offset values. For example, CADE using $N = 4$ requires 4× larger memory to store pre-computed results.

**Lookup Table Resolution:** Table III shows the impact of the tuning bits ($N$) and the offset bit width ($L$) on CADE error rate. The results show that as the value of $L$ increases (while $N$ remains a constant value), the error rate gradually reduces. However, this error reduction is smooth and stops for offsets with larger than $L = 8$ bitwidth. Furthermore, as $L$ increases, the memory size also scales accordingly with $L \times 2^{2N}$. Therefore, increasing the offset larger than 8-bits only results in degradation of CADE energy-delay product (EDP) and memory size. Since CADE error rate has a higher effect on the error rate, it is ideal to use CADE with $N = 4$ and $L = 8$. Of course, one can decide to use smaller tuning bits, if the running applications have higher error tolerate. However, increasing $L$ large than 8 has minimal effect on error rate reduction.

### D. Accuracy Vs CADE Tuning

As explained in Section II-C, CADE can achieve high accuracy through the incorporation of the tuning method. Table IV show the quality of computation in three different applications for CADE with and without tuning. We have

TABLE III
IMPACT OF THE LOOKUP TABLE BIT-WIDTH ON THE CADE ERROR RATE, EDP IMPROVEMENT AND LOOKUP TABLE SIZE ($N = 4$).

| Bit-width $L$ | | 2 | 4 | 8 | 12 | 16 |
|---|---|---|---|---|---|---|
| N=2 | ***Error Rate*** | 6.62% | 6.05% | 6.03% | 6.03% | 6.03% |
| | ***EDP Improv.*** | 14.9× | 13.6× | 12.4× | 11.1× | 9.8× |
| | ***Memory Size*** | 4B | 8B | 16B | 24B | 32B |
| N=3 | ***Error Rate*** | 6.63% | 6.00% | 4.72% | 4.72% | 4.72% |
| | ***EDP Improv.*** | 14.9× | 13.6× | 12.4× | 11.1× | 9.8× |
| | ***Memory Size*** | 16B | 32B | 64B | 96B | 128B |
| N=4 | ***Error Rate*** | 5.58% | 4.41% | 3.03% | 3.03% | 3.03% |
| | ***EDP Improv.*** | 14.9× | 13.6× | 12.4× | 11.1× | 9.8× |
| | ***Memory Size*** | 64B | 128B | 256B | 384B | 512B |


Fig. 6. Quality of computation using approximate divider on JPEG compression.

tested the quality of applications on 1000 random image from Caltech 101 [30] dataset. For the JPEG and Blur applications, PSNR was the quality metric used, while the average relative error is used to check the accuracy of the Taylor series. Both JPEG compression and Blur filter application resulted in an increase in PSNR value as $N$ increased. Comparing the average PSNR values across different data and different $N$ values, observations illustrate that JEPG compression quality computation increases from $35.1dB$ with no tuning to $50.0dB$ with $N = 4$. Similarly, for the Blur filter application, the average PSNR increases from $35.2dB$ with no tuning to $47.08dB$ with $N = 4$. In the Taylor series, for all three tested functions ($sin(x)$, $exp(x)$, and $ln(x)$), however significant decrease occurs in the average relative error when CADE configuration changes from no tuning to tuning with $N = 4$.

Figure 6 illustrates multiple images tested with no tuning and a tuning value of $N = 4$ for the Blur filter and JPEG compression. With no tuning, the Blur filter showed brighter images with lost details due to over-saturated pixels from over-approximation. JPEG images ran with no tuning, resulted in no noticeable impact on the DCT, but showed an over color saturation for certain areas resulting in a loss of detail. When tuning was introduced, the resulting images' color values were more accurately calculated to the point where the exact image and the image produced through approximation were indistinguishable to the naked eye. Overall, tuning the design has an enormous upside in relation to error reduction of different applications. The proposed tuning approach in CADE provides an opportunity to change the level of approximation at runtime, based on the running application on GPU.

## IV. CONCLUSION

In this paper, we propose a novel configurable approximate divider that efficiency divides floating point values with high accuracy. CADE removes the costly mantissa division and replaces it with a single subtraction between the two input operands mantissa. The tuning process consists of using the first $N$ bits of both input mantissas to determine the amount of error and correcting it accordingly. The CADE can be set to different levels of accuracy based on the value of $N$. Our evaluation shows that CADE is the first floating point divider which provides a new knob for GPU in order to configure the level of approximation at runtime depending on the application/user accuracy requirement.

TABLE IV
THE COMPUTATION ACCURACY OF CADE USING DIFFERENT TUNING BITS.

| Tuning Bits (N) | | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|
| **JPEG Compression** | | 35.1$dB$ | 37.4$dB$ | 42.7$dB$ | 47.7$dB$ | 50.0$dB$ |
| **Blur Filter** | | 35.2$dB$ | 39.0$dB$ | 38.7$dB$ | 41.2$dB$ | 47.8$dB$ |
| **Taylor Series** | $sin(x)$ | 1.03% | 0.51% | 0.42% | 0.24% | 0.17% |
| | $exp(x)$ | 3.54% | 1.36% | 1.22% | 0.59% | 0.21% |
| | $ln(x)$ | 1.40% | 0.84% | 0.75% | 0.33% | 0.19% |
| | **Average** | **1.81%** | **1.09%** | **0.80%** | **0.39%** | **0.19%** |


## ACKNOWLEDGEMENTS

This work was partially supported by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, and also NSF grants #1730158 and #1527034.


## REFERENCES


[1] A. Fehske *et al.*, "The global footprint of mobile communications: The ecological and economic perspective," *IEEE Communications Magazine*, vol. 49, no. 8, 2011.
[2] F. Xia *et al.*, "Internet of things," *IJCS*, vol. 25, no. 9, p. 1101, 2012.
[3] J. Gubbi *et al.*, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.
[4] B. Yao *et al.*, "Multifractal analysis of image profiles for the characterization and detection of defects in additive manufacturing," *Journal of Manufacturing Science and Engineering*, vol. 140, no. 3, p. 031014, 2018.
[5] M. Imani *et al.*, "Hierarchical hyperdimensional computing for energy efficient classification," in *DAC*, pp. 1–6, IEEE, 2018.
[6] J. Han *et al.*, "Approximate computing: An emerging paradigm for energy-efficient design," in *IEEE ETS*, pp. 1–6, IEEE, 2013.
[7] H. Amrouch *et al.*, "Towards aging-induced approximations," in *DAC*, pp. 1–6, IEEE, 2017.
[8] M. Imani *et al.*, "Approximate computing using multiple-access single-charge associative memory," *TETC*, vol. 6, no. 3, pp. 305–316, 2018.
[9] C. Liu *et al.*, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *DATE*, pp. 1–4, IEEE, 2014.
[10] M. Imani *et al.*, "Cfpu: Configurable floating point multiplier for energy-efficient computing," in *DAC*, p. 76, ACM, 2017.
[11] X. Jiao *et al.*, "Energy-efficient neural networks using approximate computation reuse," in *DATE*, pp. 1223–1228, IEEE, 2018.
[12] M. Imani *et al.*, "Masc: Ultra-low energy multiple-access single-charge tcam for approximate computing," in *IEEE/ACM DATE*, pp. 373–378, IEEE, 2017.
[13] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *DAC*, pp. 820–825, ACM, 2012.
[14] R. Ye *et al.*, "On reconfiguration-oriented approximate adder design and its application," in *ICCAD*, pp. 48–54, IEEE, 2013.
[15] Z. Yang *et al.*, "Approximate xor/xnor-based adders for inexact computing," in *Nanotechnology (IEEE-NANO), 2013 13th IEEE Conference on*, pp. 690–693, IEEE, 2013.
[16] S. Hashemi *et al.*, "Drum: A dynamic range unbiased multiplier for approximate applications," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 418–425, IEEE Press, 2015.
[17] P. Kulkarni *et al.*, "Trading accuracy for power with an underdesigned multiplier architecture," in *IVLSI*, pp. 346–351, IEEE, 2011.
[18] C.-H. Lin and C. Lin, "High accuracy approximate multiplier with error correction," in *ICCD*, pp. 33–38, IEEE, 2013.
[19] G. Bradski and A. Kaehler, "Opencv," *Dr. Dobbs journal of software tools*, vol. 3, 2000.
[20] W. Liu and A. Nannarelli, "Power efficient division and square root unit," *IEEE Transactions on Computers*, vol. 61, no. 8, pp. 1059–1070, 2012.
[21] S. Hashemi, R. Bahar, and S. Reda, "A low-power dynamic divider for approximate applications," in *Proceedings of the 53rd Annual Design Automation Conference*, p. 105, ACM, 2016.
[22] S. Vahdat *et al.*, "Truncapp: A truncation-based approximate divider for energy efficient dsp applications," in *DATE*, pp. 1639–1642, IEEE, 2017.
[23] L. Chen *et al.*, "Design of approximate unsigned integer non-restoring divider for inexact computing," in *GLSVLSI*, pp. 51–56, ACM, 2015.
[24] M. Imani *et al.*, "Ultra-efficient processing in-memory for data intensive applications," in *DAC*, p. 6, ACM, 2017.
[25] M. Imani *et al.*, "Acam: Approximate computing based on adaptive associative memory with online learning," in *ISLPED*, pp. 162–167, ACM, 2016.
[26] S. Jain *et al.*, "Binary division algorithm and high speed deconvolution algorithm (based on ancient indian vedic mathematics)," in *ECTI-CON*, pp. 1–5, IEEE, 2014.
[27] Ubal *et al.*, "Multi2sim: a simulation framework for cpu-gpu computing," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pp. 335–344, ACM, 2012.
[28] "Aflopoco [online]. available:http://flopoco.gforge.inria.fr/,"
[29] R. Zendegani *et al.*, "Seerad: A high speed yet energy-efficient rounding-based approximate divider," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, pp. 1481–1484, EDA Consortium, 2016.
[30] L. Fei-Fei *et al.*, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," *Computer vision and Image understanding*, vol. 106, no. 1, pp. 59–70, 2007.