Approximate Computing Technique Using Memoization and Simplified Multiplication

Yoshinori ONO† and Kimiyoshi USAMI‡*

† *Graduate School of Engineering and Science, Shibaura Institute of Technology*
‡ *Department of Information Science and Engineering, Shibaura Institute of Technology*
* *Research Center for Green Innovation, Shibaura Institute of Technology*
E-mail: †, ‡ {ma19013, usami}@shibaura-it.ac.jp*

Abstract

In embedded systems, approximate computing can strongly promote reduction of execution time and energy consumption in exchange for some output errors. We focused on "Fuzzy memoization", which is one of the approximate computing techniques. We improved it by using simplifying multiplication. By using this approach, we have developed a novel technique to reduce execution time and energy consumption while keeping output precision. Then, we applied it to grayscale filters on the Zynq system that contains ARM-based processor and field-programmable gate array (FPGA). Evaluation results from the implemented system showed that our proposed technique can reduce the execution time by up to 28% and reduce the energy consumption by 11% in spite of very high-quality output images.

Keywords: Approximate Computing, Image Processing, Field Programmable Gate Array (FPGA)

1. Introduction

Approximate computing is known as a method of reducing execution time or energy consumption by allowing inexact computing. Media processing represented by image processing is acceptable even for incorrect results [1]. Therefore, this method can achieve short execution time by allowing to approximate an accurate operation using this property.

In this method, it is important to improve the trade-off relation between error rate and execution time or energy consumption [2]. If error rate is high, we may not get the right information from the output because of unacceptable loss of quality. Therefore, it is required to maximize its gain while keeping the quality.

Fuzzy memoization [3] is one of the approximate computing techniques which is based on instruction memoization [4]. Both memoizations store input and output data for a process as an entry in the memo table, and reuse it to reduce execution time by skipping the original process. Instruction memoization reuses entries only for the same input. However, fuzzy memoization reuses similar input. Hence, adopting fuzzy memoization can reduce execution time more. This technique was applied to various systems [5-7].

Figure 1 shows the process of fuzzy memoization. x is an input and R is an output in the whole system. x is also an input to the Memo Table at the same time with original formula $f(x)$. If there is an entry within the tolerance for x in the table, Memo Table puts the result R_m in the entry and a hit signal is given to the multiplexer (MUX). Then, the MUX puts R_m as the result of the whole system. At this time, the calculation by the original formula $f(x)$ is forcibly terminated and the processing for the next input is started. On the other hand, if the input is not in the range of the tolerance for x, the system calculates the precise result R_x with $f(x)$ to the last and R_x is the result of the system. At the same time, a pair of x and R_x is stored in a table as an entry. The more the entries are used, the shorter execution time can be. However, the search time in the memo table needs to be shorter than the calculation time of the original formula.

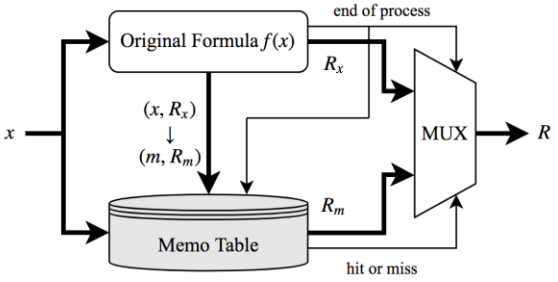


Figure 1: Fuzzy memoization system

However, fuzzy memoization has problems with error rate. The size of the table and tolerance level varies to achieve the maximum gain. In this paper, tolerance means the range of input to hit for an entry. If tolerance is large, error rate increases.

To tackle these problems, we propose an approach to use fuzzy memoization and a simplified calculation. Through an application to grayscale filters, we show effectiveness of the proposed approach.

The paper is organized as follows. Section 2 describes the proposed approach. Section 3 describes the evaluation setup. Section 4 and 5 present results and discussions, respectively. Section 6 concludes this paper.

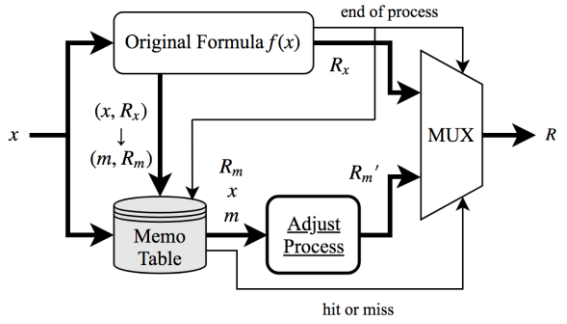
2. Proposed approach

2.1. Proposed system

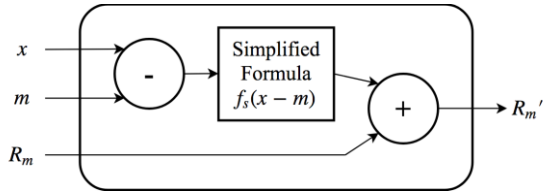
We propose an approach to improve the output quality of the fuzzy memoization while minimizing performance penalties.

Figure 2 shows the process of proposed system. When an entry hits, the proposed technique recalculates the output of the entry to make it closer to the exact value. In advance, we simplified the original formula (e.g. conversion to grayscale). If hit, this technique calculates the value by a simplified formula that takes input as difference between current inputs and entry's inputs. Then, it becomes a more accurate value by adding this value and entry's result. In other words, the proposed system reduces the error rate by adjusted the output of an entry by a simplified formula.

In order to obtain the above system, we decided to put an adjust process between the memo table and the MUX of fuzzy memoization. In this process, m is the input of the entry which is hit and R_m is the result of the entry pair with m. The difference between x and m is the variable for the simplified formula f_s. The result of f_s is added to the R_m to earn more accurate value. However, the execution time on search for the R_m and adjust process must be shorter than an original formula.



(a) Overall



(b) Adjust Process

Figure 2: Proposed system

2.2. Application

We adopted for converting color images to grayscale as an application in testing the proposed system. The size of these images was 512px x 512px. We used shift operation in considering simplification of conversion's multiplication for hardware or FPGA to approximate the multiplication. Assuming that the arithmetic right shift is $\gg$, f and f_s are as follows.

$$f(x_r, x_g, x_b) = 0.299x_r + 0.587x_g + 0.114x_b$$

$$f_s(x_r, x_g, x_b) = x_r \gg 2 + x_g \gg 3 + x_g \gg 1 + x_b \gg 3$$

2.3. Implementation

We chose a Zynq processor to implement the image processing unit in a FPGA and to facilitate system development. The Zynq processor is an integrated SoC chip in which an ARM CPU and a Xilinx FPGA are embedded. It is ideal for configuring a system that combines software and hardware. We set the operating frequency of the FPGA part to 50MHz. In addition, we implemented the modules so that they can operate in parallel as much as possible.

3. Evaluation setup

We evaluated execution time in part of FPGA, energy consumption and output error rate. For the output error rate, we used PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity), which can quantitatively evaluate the image quality. The data are the averages when processing all images in this experiment. PSNR greater than 30dB is generally considered acceptable from user perspective.

The table size (maximum number of entries that can be stored) can be set 1, 2, 4, 8, 16, 32, 64 and 128. The tolerance can be set from ± 1 to ± 30.

4. Results

4.1. Output accuracy

Figure 3 represents output images for three different types systems: the precise computing system, the fuzzy memoization system and the proposed system. The memo table size of these system was set to 128 and tolerance was set to ± 30. The image output with the fuzzy memoization system had noise in a horizontal direction. However, the image output with the proposed system did not have the noise and was almost equal to the precise computed image.

The PSNR was improved than fuzzy memoization by using the proposed system when table size was larger or tolerance was bigger. The PSNR of the proposed system never fell below 45db. However, depending on the table size, the PSNR of fuzzy memoization dropped to less than 30db when tolerance was bigger than 24.

The SSIM was also improved than fuzzy memoization by using the proposed system. The SSIM of the proposed system is above 0.99 in all settings. However, the SSIM of fuzzy memoization dropped to less than that of the proposed system.

4.2. Execution time

The larger the table size or tolerance, the shorter the execution time. The proposed system was able to save time more than the precise computing system for all table sizes and tolerance. However, fuzzy memoization could reduce the time further. The reduction rate of the proposed system was about half of fuzzy memoization.

Precise computing Fuzzy memoization Proposed system

Figure 3: Grayscale Images

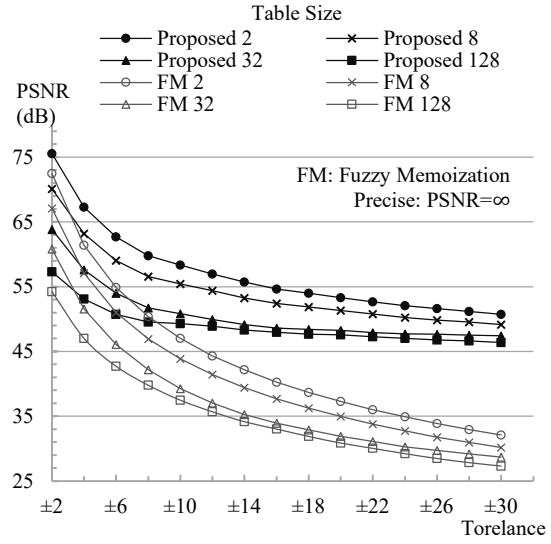


Figure 4: PSNR

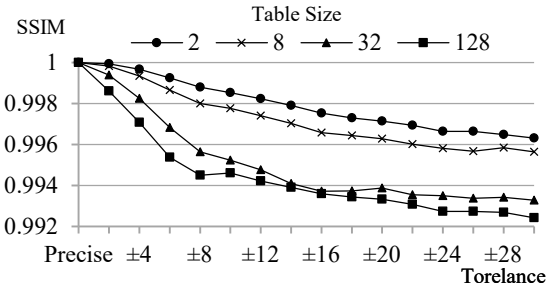


Figure 5: SSIM - Proposed System

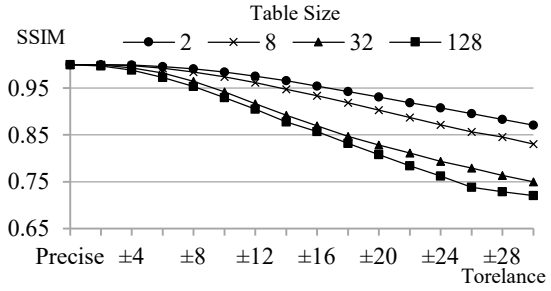


Figure 6: SSIM - Fuzzy Memoization

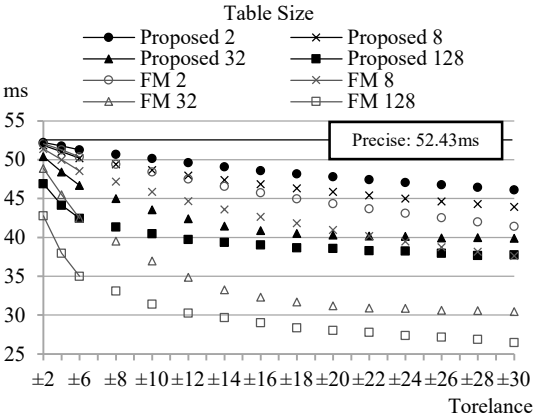


Figure 7: Execution Time

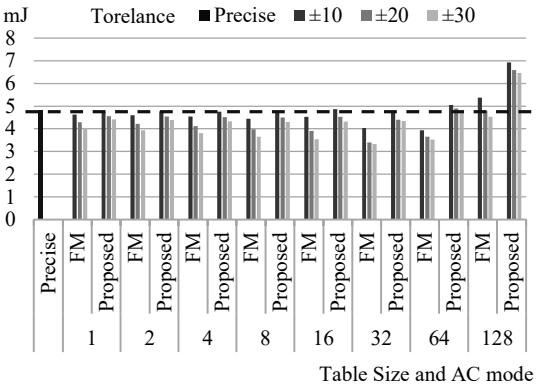


Figure 8: Energy Consumption

4.3. Energy consumption

When the memo table size was between 1 to 32, the proposed system and fuzzy memoization reduced energy consumption compared to the precise computing system. However, there was no improvement on both systems when the table size was greater than 32. The shortening rate of the proposed system was about 1/2 to 1/3 of fuzzy memoization.

Table 1: Minimum, maximum and average of each evaluation

	Precise Computing	Fuzzy Memoization			Proposed		
		Minimum	Maximum	Average	Minimum	Maximum	Average
PSNR	∞	23.33 dB	75.11 dB	40.10 dB	46.40 dB	78.38 dB	53.47 dB
SSIM	1.0	0.7171	0.9999	0.9087	0.9922	0.9999	0.9962
Execution Time (Improvement)	52.43 ms	26.50 ms (49%)	52.20 ms (0.44%)	40.12 ms (23%)	37.70 ms (28%)	52.30 ms (0.24%)	45.40 ms (13%)
Energy Consumption (Improvement)	4.82 mJ	3.32 mJ (31%)	5.37 mJ (-11%)	4.12 mJ (15%)	4.30 mJ (11%)	6.92 mJ (-44%)	4.86 mJ (-0.83%)

4.4. Minimum, maximum and average

Table 1 shows the minimum, maximum and average value of PSNR, SSIM, execution time and energy consumption. In addition, execution time and energy consumption also indicate improvement rates when compared with precise computing. The proposed system is effective for PSNR and SSIM, and fuzzy memoization is effective for execution time and energy consumption.

5. Discussions

We achieved reduction of execution time and energy consumption while keeping accuracy by the proposed system. Comparing the output image, PSNR and SSIM of fuzzy memoization, the accuracy was improved significantly on the proposed system. In addition, our system achieved to reduce execution time and energy consumption compared to the precise computed system, although fuzzy memorization reduced more. By using this system, it is possible to meet the demand for reduction of execution time and energy consumption even in image processing of embedded systems that requires quality.

We had to be careful to set the memo table to an appropriate size. As we confirmed the implementation results, the larger the table size, the more lookup tables on the FPGA were used. This has increased power consumption, and has had a greater impact than the reduction in energy consumption due to reduced execution time. From figure 8, we think that the table size should be kept below 32.

As a further improvement, it should increase the reuse rate of the memo. By promoting the reuse of memos, execution time is reduced because a calculation of original formula is skipped. This system should implement a more efficient algorithm for updating the memo tables.

6. Conclusion

We proposed an approach to combine two approximate computing techniques of fuzzy memorization and simplification of multiplication by shift operation. By constructing a system to implement the proposed systems, we demonstrated that it reduces execution time and energy consumption while maintain the accuracy.

As the future work, the algorithm to update the memo entries should be studied to further reduce execution time and energy and skipping more the original formula.

Acknowledgment

This presentation was supported by SIT Research Center for Green Innovation.

References

[1] Sparsh Mittal, “A Survey of Techniques for Approximate Computing”, ACM Computing Surveys, Vol. 48, No. 4, Article 62, March 2016.

[2] M. A. Breuer, "Multi-media applications and imprecise computation", 8th Euromicro Conference on Digital System Design (DSD'05), IEEE, pp. 2-7, September. 2005.

[3] C. Alvarez, J. Corbal, M. Valero, “Fuzzy Memoization for Floating-Point Multimedia Applications”, IEEE Transactions on Computers, Vol. 54, No.7, July 2005.

[4] D. Citron, D. Feitelson, L. Rudolph, “Accelerating Multi-Media Processing by Implementing Memoing in Multiplication and Division Units”, Proc. Eighth Int’l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS VIII), ACM, pp. 252-261, October 1998.

[5] A. Rahimi, L. Benini, R. K. Gupta. “Spatial memoization: Concurrent instruction reuse to correct timing errors in SIMD architectures”, IEEE Transactions on Circuits and Systems II: Express Briefs, Vol. 60, Issue. 12, pp. 847–851, December 2013.

[6] M. Samadi, D. A. Jamshidi, J. Lee, S. Mahlke, “Paraprox: Pattern-based approximation for data parallel applications”, ACM SIGARCH Computer Architecture News (ASPLOS '14), Vol. 42, Issue 1, pp. 35–50, March 2014.

[7] A. K. Mishra, R. Barik, S. Paul, “iACT: A Software-Hardware Framework for Understanding the Scope of Approximate Computing”, In Workshop on approximate computing across the system stack (WACAS ’14), ACM, March 2014.