Stockham FFT acceleration with Processing-in-Memory

Byoung Jin Kim, Tae Yang Jeong, and Eui-Young Chung
School of Electrical and Electronic Engineering, Yonsei University
50 Yonsei-ro Seodaemun-gu, Seoul, Korea
E-mail: bryankim@yonsei.ac.kr, drthvbfg@yonsei.ac.kr, eychung@yonsei.ac.kr

Abstract

Stockham, one of the most widely used FFT algorithms, does not need reversal permutation, but memory access patterns are unpredictable. Because of the cache-unfriendly memory access patterns, the performance of Stockham could be degraded. Processing-in-Memory approach could alleviate the memory access overhead and allow Stockham algorithm to be accelerated. We evaluated the effectiveness of Processing-in-Memory with Gem5 full system simulator. As a result, the maximum performance gain was 4.22 times faster than CPU only environment.

Keywords: Processing in Memory, Accelerator, FFT, Stockham Algorithm

1. Introduction

Fast Fourier Transform(FFT) is used for various fields including engineering, scientific computing, and financial technology. There are many algorithms for FFT, but Stockham is one of them. Stockham iteratively performs FFT operation without reversal permutation[1]. However, the memory access patterns of Stockham are not sequential and hard to predict. The performance of Stockham is related to not only processing throughput but also memory bandwidth.

Processing-in-Memory(PIM) is a approach which has processing elements near memory. Recently, there are numerous studies on PIM for accelerating applications[2][3]. Host processor could offload some computations into PIM-side processing elements. The processing element of PIM does not pass through the host processor's cache, PIM prevents unnecessary cache thrashing. With parallel processing elements, PIM takes full advantage of the memory bandwidth.

In this paper, we present a novel PIM architecture for Stockham FFT. Our architecture is evaluated with Gem5[4] full system simulator, and the results show that Stockham FFT could be processed efficiently with PIM approach.

2. Stockham FFT Algorithm

Figure 1 shows the iterate execution flow and the memory access patterns of Stockham FFT algorithm. In this figure, the input size of FFT is set to 8. Increasing the input size, Stockham runs more iterations and the memory access pattern would be more complicated.

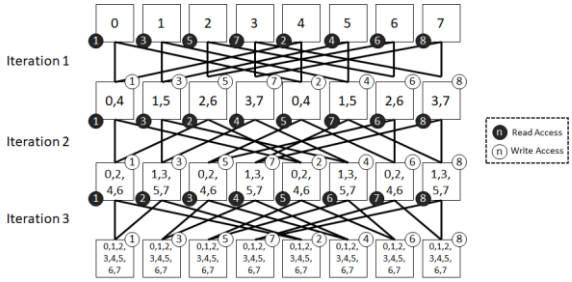

Figure 1. Memory access patterns of Stockham

Each iteration of Stockham could not be parallelized, on the other hand, the operations within an iteration could be parallelly performed. However, it is not easy to process in parallel, because the memory access pattern changes every iteration.

The computation of each operation consists of a cosine function, a sine function for obtaining the real and imaginary part from a complex number, and multiplying between complex numbers, and some simple arithmetic operations. These operations could be heavy for general purpose processors.

3. PIM architecture for Stockham FFT

Proposed PIM architecture is shown in figure 2. For processing near memory, we added some control blocks and processing element into the conventional memory controller.

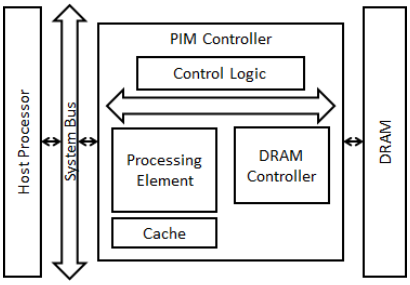

Figure 2. Proposed PIM architecture

In this system, the main memory is PIM which contains processing elements for accelerating Stockham FFT. For offloading PIM computation to PIM controller, we implemented a Linux device driver. Driver passes PIM requests containing the addresses of source and destination and the size of data.

To minimize the size of the processing element, we only parallelized a single operation within an iteration. Cosine function and sine function, which are the heaviest operations of Stockham algorithm, are parallelly processed and some double-precision floating point ALUs are added. The number of used units for the processing element is shown in Table 1.

Table 1. The number of each unit consists of the processing element

Cosine logic	1
Sine logic	1
Double-precision floating point ALU	6
Integer ALU	1

4. Experiment

Our evaluation is based on Gem5 full system simulator. We implemented the proposed PIM architecture into the conventional DRAM controller. Detailed simulation configuration is shown in Table 2. We assumed several configurations such as ALU cycles.

Table 2. Simulation Environment

Host processor	ARM Cortex A15 @ 1GHz
Cache	L1-I/Dcache: 2-way 16KB,
DRAM	DDR4-2400, 17-17-17
Double-precision floating point ALU cycles	4(ADD, SUB), 6(MUL), 25(DIV), 400(Sin), 420(Cos)
Integer ALU cycles	1(ADD, SUB), 3(MUL), 20(DIV)

5. Experimental results

Our experimental results are shown in Figure 3. We evaluated the overall performance of Stockham FFT in PIM, compared with CPU. In future work, we plan to compare PIM against GPGPU.

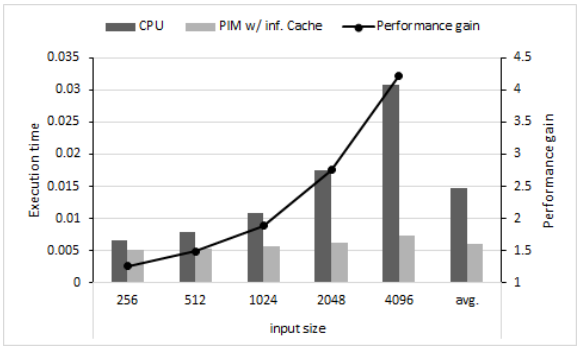

Figure 3. Simulation results

With parallel processing, PIM predominates over all input sizes. When input size is increasing, especially, PIM performance gain increases. Because of the more complicated memory access patterns, PIM is more powerful in large input sizes. The result shows, in short, 4.22x of maximum performance gain, and 2.32x of average performance gain.

6. Conclusion

In this paper, we proposed a novel PIM architecture for Stockham FFT. Offloading FFT computations into PIM-side, proposed PIM achieves 2.32x average performance gain against CPU. We will further investigate PIM architecture for other applications.

Acknowledgement

This work was supported by the Institute of BioMed-IT, Energy-IT, and Smart-IT Technology (BEST), a Brain Korea 21 plus program, Yonsei University, and by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (2016R1A2B4011799)

References

[1] D. Brandon Lloyd, Chas Boyd, Naga Govindaraju, “Fast Computation of General Fourier Transforms on GPUs”, IEEE International Conference on Multimedia and Expo, Apr. 2008

[2] Paulo C. Santos, Geraldo F. Oliveria, Diego G. Tome, Marco A. Z. Alves, Eduardo C. Almeida, Luigi Carro, “Operand Size Reconfiguration for Big Data Processing in Memory”, Design, Automation & Test in Europe Conference & Exhibition(DATE), Mar. 2017

[3] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, Kiyoung Choi, “A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing”, ACM/IEEE

42nd Annual International Symposium on Computer Architecture(ISCA), June. 2015

[4] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, “The Gem5 Simulator”, SIGARCH Comput. Archit. News, vol. 39, no. 2, pp. 1–7, Aug. 2011