Low-Power High-Speed CNN Accelerator with Matrix Reodering Techniques for Small Footprint Memory Access

Hoseong Kim and Daejin Park*
School of Electronics Engineering, Kyungpook National University, Daegu, Korea
*Correspondence: boltanut@knu.ac.kr

***Abstract*—In the evolution of artificial intelligence (AI), large - scale data volumes are needed to best represent the information. This dataset is expressed by a matrix or tensor. When data volumes increase, the operational throughput of data processing increases, exponentially. However, this is not acceptable for computational devices, such as mobile devices, that are expected to low power consumption and high-performance. As the demands for embedded chip devices in AI software have increased, low-power and high-performance modules for AI software are needed to ensure efficient matrix or tensor data processing. Commonly, as performance enhances, power consumption also increases. In this point, a new matrix operation system is essential to ensure high performance and low-power consumption for efficient matrix data processing. In the matrix operation property, large amounts of computational operation can be reduced and sped up through parallelized data processing. In this paper, to determine the advantages of parallelized matrix operation in terms of performance and power consumption, hardware (ALU and memory) module chip equipped with a parallel operation structure and algorithm is designed to reduce data volumes and operational load. By implementing this module by Verilog, with a multicore processor, the processing speed and power consumption are able to compare with the existing sequential processing method of a single-core structure processor [1][2]. We conducted an experiment to check the operation and performance of a parallelized module chip. The data are stored in memory to compression form through CNN algorithm software, and this is useful in the operation of the matrix arithmetic. This matrix arithmetic operation load is also reduced in support of the CNN algorithm software. The experimental results most likely confirm that the designed quad-core processor embedded CNN algorithm increased its speed to n*log n scale time complexity and that the memory usage rate, memory access footprint, and power consumption remarkably decreased.**

Keywords—Matrix or Tensor data processing; parallelized processing; multicore processor; Verilog; Convolutional neural network; Pruning alogrithm; Deep Compression alogrithm; arithmetic algorithm;

I. INTRODUCTION

A large amount of information data takes the form of a matrix or tensor data structure. A matrix or tensor has various information forms and complex data types. However, these multidimensional matrix processes demand much time and high energy because sequential computational structure processing usually processes one word, at a time, and has high iterative data access for operation. As the data volumes increases, the operational throughput also increases. In general, the $n \times n$ matrix multiplication expressed by C programing has n^3 time complexity due to the triple loop [3]. In this matrix product sequence, some properties of matrix computation can be used to enhance the operational performance.

Independence and iteration are major properties of the matrix arithmetic operation. For the matrix product, the result matrix element is not dependent on the sequence of operation, and each operand matrix's elements are iteratively accessed in the same times. Matrix arithmetic operation can be carried out by means of these accessed data [4]. From these characteristics, the parallelized data processing hardware module can be implemented. In this structure, only one loop is needed for the matrix's operation, so the time complexity of the parallelized method's matrix product can be reduced as a first order linear function (Fig. 4.). Also, using multicore structure allows researchers to easily deal with enormous and continuous matrix data flows.

In matrix or tensor data processing, all data elements do not have an impact on effective results. That is, it is not necessary to perfect the matrix product. Although some of the data elements are altered to have a random value to some of the small extent, the total operation outputs of information process will most likely be unchanged. These unimpacted data can be ignored to zero. This is called pruning [5]. After initiation the pruning method, these zero data do not require memory storage. So, matrix data compression first occurs in the pruning process. However, this reduced matrix data storage does not represent the original matrix structure due to the modified array index. In this point, a new matrix data expression method must be introduced to locate a matrix's index information. Simply, the data value and index information can be stored in memory. Furthermore, these element values are more compressed by its range. The sampling and selection of representative values can be quantized from floating point to small range bit code. So, a new data expression approach can be introduced for efficient data memory storage [6].

In the data flow level, a lot of data elements execute product operation in a parallel processing way from hardware module implemented in this paper. The data type inflowed by the input gate module is generally in the floating-point format, which is the real number. So, data throughput is heavier and more complex than the integer type number. For high speed and low power, the data arithmetic cell is needed for more efficient operational sequences and algorithms. In this point, an

appropriate and efficient arithmetic cell for processing the floating point is introduced.

The first objective of this paper is to verify the advantages of this implemented parallel processing matrix multiplication module in terms of execution time, less power consumption as compared to the sequential processing module. The second objective is to verify the data compression efficiency and memory usage in the matrix data loaded. We demonstrate that this system implements the parallel hardware and CNN software algorithm appropriately in operating and check the effective performance in a point of data process improvement, power consumption reduction, memory usage compression, and optimal memory access footprint.

II. Proposed Method

The hardware structure and the algorithm for matrix operation in parallel processing are based on the repeatability and same number of times access of each operand element. In the matrix product, each resulting element can be reconstructed via the first operand matrix's column and second operand matrix's row multiplication, namely, the sum of the partial matrix state (Fig. 1.). Different from the general inner products about the result matrix's index, these partial states form the template matrix with the same result as to the matrix dimension, but these templates represent only a part the of result matrix. These partial matrices, called template matrices, form a total of n parts, which are same as to the number of first operand matrix column - second operand matrix row.

For example, if $m \times n$ and $n \times p$ matrices operate product, this operation creates the first to nth column-row product template matrices. And then, these template matrices can build up the result matrix. Actually, when using padding for easy operation and expression, the matrix format can be fixed to $n \times n$ dimension. Although the parallelized product's hardware needs more ALUs than original computer structure, the arithmetic cell ALUs in this hardware for MAC-multiply and accumulation-operation is much smaller and light due to focused on only the fast and simple operation method. As a result, the speed of matrix product is much higher, and the time complexity is reduced to the first order linear function of n through this parallelized matrix product operation (Fig. 2.). In implementing the hardware for the multicore structure, the hardware module chip can easily process the multidimension and large matrix data elements more than the number of hardware arithmetic cells with very high-speed execution time (Fig. 3.). Using the scheduling algorithm conducted as the control logic for core efficient and powerful usage, the total speed and power consumption can be reduced much less than the operation of a single core matrix product's hardware chip. This can be done by implementing a parallel processing matrix product module by Verilog using Intel's FPGA design software and checking the module operation behavior using Modelsim.

The operand matrices can be optimized for volume reduction induced by the pruning algorithm (Fig. 5.). Actually, all data in the matrix do not have an impact on the result. Only some data significantly impact the result matrix. The pruning step can retain significant data to affect and delete redundant data. For this, the threshold value is induced, which can make a criterion whether the data pruning. If the matrix data element is less than the threshold data value, all element values satisfied with the threshold condition should be changed as zero. This pruning is fulfilled until the pruning matrix attains the desired sparsity level. The important aspect of the pruning algorithm is that a well-fitting selection threshold value and sparsity criterion represents no loss result data output. So, threshold value selection is needed in the training steps to improve the precision of the pruning algorithm, and the sparsity criterion is used to normalize of the pruned matrix.

After pruning, the compressed matrix array modifies the original data dimension structure by neglecting the pruned zero elements. These compressed matrices must be stored in memory with location information in order to preserve the original matrix information. Various representative formats are induced to ensure the sparse matrix expression for memory storage. These representations can be matrix data that are more compressed than COO (coordinate format) representation. In this paper, we used the CSR (compressed sparse row) format for matrix expression, and we implemented the transposed operation algorithm for matrix product (Fig. 6.). In the sparse matrix, there are small numbers of remaining meaningful data values, so these values can be compressed even more using quantization algorithm (Fig. 7.). The quantization method is all about sampling the representative value; it can be described as data sharing and encoding the representative values to the code table. Quantization and pruning can compress the matrix for the very high-compression rate, and the output result accuracy can be preserved, whereas only useful data values are compressed via the code table in the small bits used.

We compressed the pruned matrix data as the code table, and these encoded matrix data are stored in memory. The hardware ALU only calculates the remaining data in a compressed matrix representation (Fig. 9.), and then it stores and accumulates the operated values in the right index of the result matrix register (Fig. 10.). Only the remaining data must be operated, so it does not matter where the index location data are operated. In the arithmetic cell, the bit data are translated as actual data values from the code table. So, we need to decode the algorithm in the data code. The data code is different when it comes to what data are quantized, what group of values are sampled, and what data are selected as representative. So, we update the decode hardware every data code table is exchanged. Also, the code data has finite value range, so we can reduce the precision of actual data. This can be possible that FP32 and FP64 format must not be kept as the regulation (Fig. 8.). This data decoder and fluctuated fixed range format can possibly accelerate the matrix multiplication.

III. Conclusion

The parallel processing method of matrix multiplication confirms that the speed performance obviously improves, and it can be easily confirmed by Modelsim simulation. The power consumption of total operation is much smaller than the sequential computing structure because optimized algorithms are used for data storage and processing. However, the average power consumption is much higher, due to many parallel ALU usages and much smaller execution time. In this paper, the main processes of matrix operation, such as intensive data access and

floating operation, are greatly reduced; this can make it possible to the lower power for matrix multiplication operation. Compressed matrices can be usually operated in the register block. The SRAM cache's energy consumption for data access, (i.e. 5pJ) is very small compared to that of DRAM, (i.e. 64pJ). Also, the main floating operation is reduced in small bits; this is much less energy consumption compared with the 32-bit floating multiplication, (i.e. 3.7pJ). Therefore, linear data throughput increase may possibly result in much less time consumption in very large-scale dataset processing (Fig. 11.). Also, memory reordering techniques can be possible with regard to very small memory usage for large data. Consequently, high-efficient data processing of the multidimension matrix operation in restricted devices, such as embedded system, can be possible.

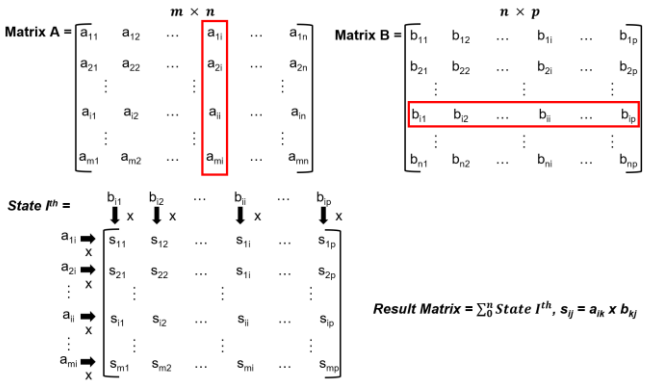

Fig. 1. Structure of parallelized matrix product.

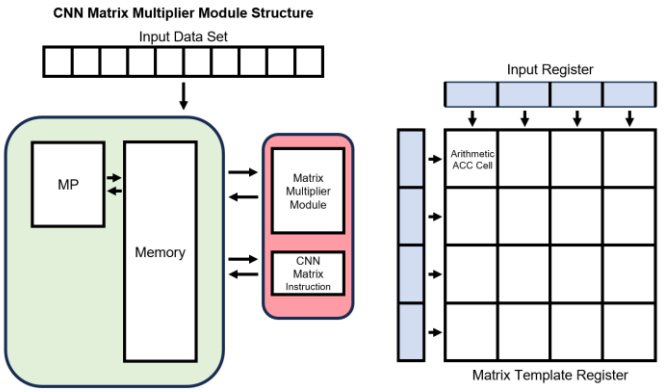

Fig. 2. Hardware module of parallelized matrix product.

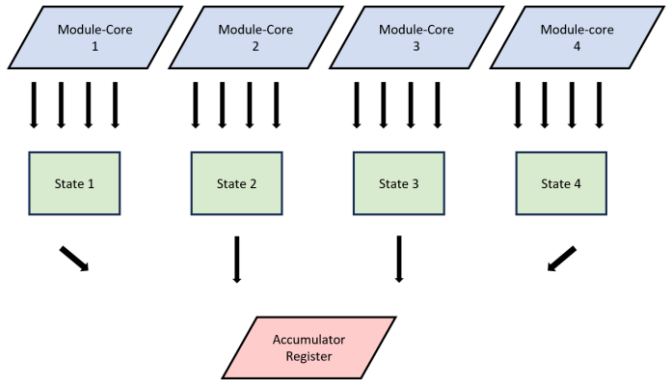

Fig. 3. Parallelized matrix multiplication data flow diagram.

Algorithm sequential matrix multiplication (Matrix_A, Matrix_B, Matrix_C)
Multiply Matrix_A and Matrix_B by sequential processing way
Pre Matrix_A and Matrix_B are n by n matrices
Post matrices multiplied--result in Matrix_C
1 for (i = 0 ; i < n ; i ++)
1 for (j = 0; j < n ; j ++)
1 for (k =0 ; k < n ; k ++)
1 calculate sum of Matrix_A[i][k]*Matrix_B[k][j]
2 store sum in Matrix_C[i][j]
2 end loop
2 end loop
2 end loop
3 return
end sequential matrix multiplication

Time complexity of sequential processing : $O(n^3)$

Algorithm parallel matrix multiplication (Matrix_A, Matrix_B, Matrix_C)
Multiply Matrix_A and Matrix_B by parallel processing way
Pre Matrix_A and Matrix_B are n by n matrices
result Matrix_C is sum of temporary state matrix, S, n by n
Post matrices multiplied--result in Matrix_C
1 for (i = 0 ; i < n ; i ++)
1 calculate temporary nth state matrix S[i][j] = Matrix_A[i][n]*Matrix_B[n][j]
2 accumulate state matrix to result Matrix_C
2 end loop
3 return
end parallel matrix multiplication

Time complexity of parallel processing : $O(n)$

Fig. 4. Pseudocode and time complexity.

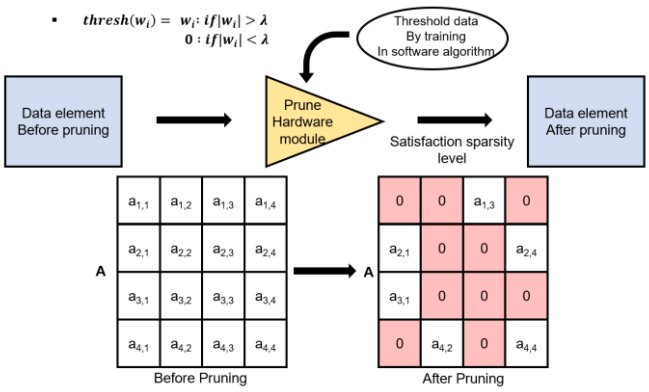

Fig. 5. Pruning algorithm.

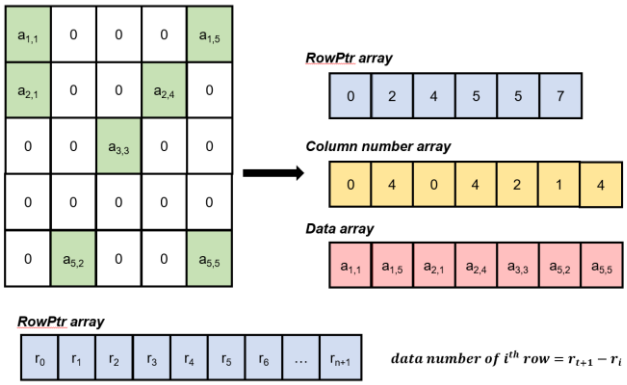

Fig. 6. CSR compression diagram.

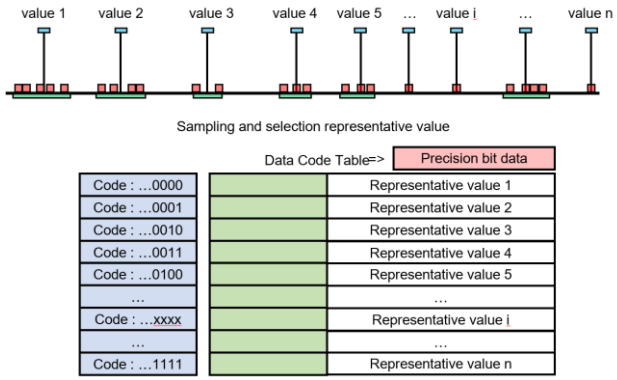

Fig. 7. Quantization

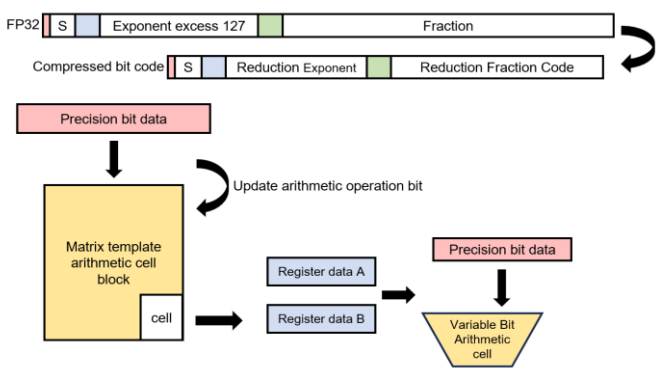

Fig. 8. Arithmetic cell variable bit operation diagram.

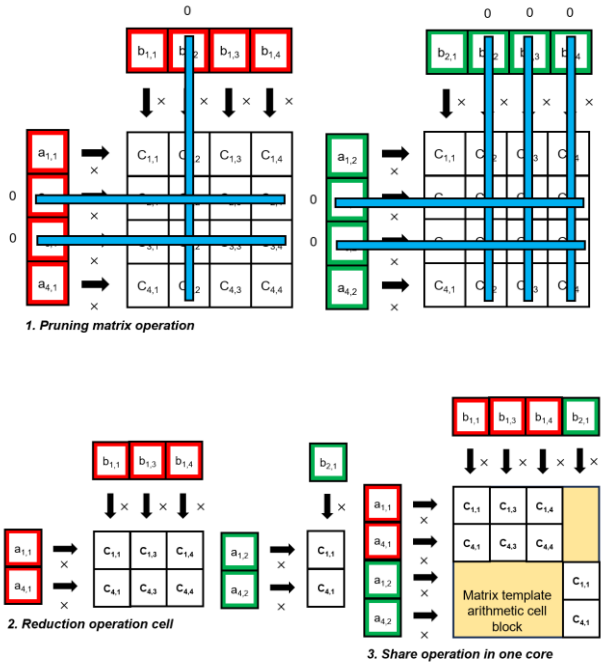

Fig. 9. Matrix operation core sharing of pruning matrix

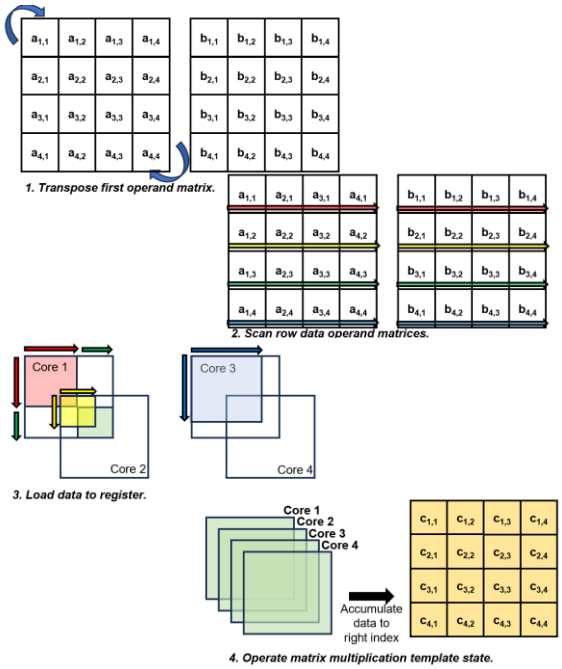

Fig. 10. Matrix product module process diagram.

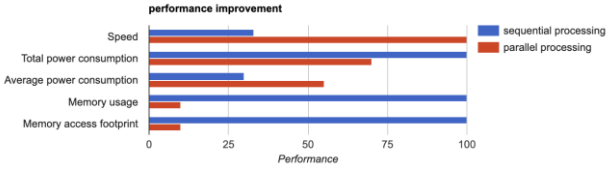

Fig. 11. Performance comparison statistics diagram.

ACKNOWLEDGMENT

This study was supported by the BK21 FOUR project (4199990113966), the Basic Science Research Program (NRF-2018R1A6A1A03025109, 10%), (NRF-2022R1I1A3069260, 10%) through the National Research Foundation of Korea (NRF) funded by the Ministry of Education. This work was partly supported by an Institute of Information and communications Technology Planning and Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2022-0-01170, PIM Semiconductor Design Research Center, 30%) and (No. RS-2023-00228970, Development of Flexible SW-HW Conjunctive Solution for On-edge Self-supervised Learning, 50%). The EDA tool was supported by the IC Design Education Center (IDEC), Korea.

REFERENCES

[1] Samir Palnitkar, “Verilog HDL, A guide to digital design and synthesis”, Prentice Hall, ISBN 89-7283-501-3

[2] Lech Jowiak, Yahya Jan, “Hardware Multi-processor Design for Highly-Demanding Applications”, 2013 Tenth International Conference on Information Technology: New Generations (ITNG)

[3] Richard F. Gilberg, Behrouz A. Forouzan, “Data structures : A Pseudocode Approach with C, 2nd endition”, Cengage India, ISBN 978-813150314

[4] David C. Lay, “Linear Algebra and Its Applications, Global edition, 6th edition”, Pearson, ISBN 978-1292351216

[5] Song Han, Jeff Pool, John Tran, William Dally, “Learning both weights and Connections for Efficient Neural Network”, Advances in Neural information Processing Systems(NIPS), Pages 1135-1143, 2015.

[6] Song Han, Huizi Mao, William J Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantizaion and Huffman Coding”, International Conference on Learning ===Represention(ICLR’16 best paper award)