2024 Asia Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC) | 979-8-3503-6733-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/APSIPAASC63619.2025.10849235

Transformer Attention Matrix Multiplication Design using 4×4 Systolic Arrays

Muhammad Sayyid Afif[‡], Infall Syafalni[‡§], Nana Sutisna[‡§], and Trio Adiono[‡§]
[‡]School of Electrical Engineering and Informatics, Bandung Institute of Technology, Indonesia
[§]University Center of Excellence on Microelectronics, Bandung Institute of Technology, Indonesia

***Abstract*—Transformers have revolutionized natural language processing (NLP) with their multihead attention mechanism, enabling efficient parallel processing of text data. This paper enhances Transformer performance using a systolic array architecture for matrix multiplication. We optimize the computation process by leveraging parallel partitioning to accelerate self and multi-head attention mechanisms within Transformer models. Utilizing the BERT model, we partition its 768-dimensional vectors and process to make multiple smaller 4×4 structures. The experiments were conducted using software simulations and FPGA Zybo 7020 hardware design synthesis. The primary objective is to improve computational throughput and resource efficiency, leading to the development of lightweight and high-performance Transformer models. This study highlights the potential of parallel partitioning with systolic arrays in matrices multiplication advancing the Transformer's computation processing tasks. Our experiment simulates the parallel partition and synthesizes the hardware architecture design. By comparing the conventional nested loop and parallel partition method, we find a 66.9% speed increase, with maximum frequencies of 91.7. Our findings contribute to optimizing Transformer models for various applications, emphasizing the role of hardware acceleration.**

***Index Terms*—Transformer, Multihead Attention, Accelerator, Systolic Array, Natural Language Processing.**

I. Introduction

TRANSFORMER have revolutionized the field of natural language processing. The Transformer model introduced a novel mechanism called multihead attention, which has proven to be highly effective in capturing contextual relationships and dependencies in text data [1]. Using multihead attention, the Transformer model can process text in parallel without relying on sequential operations like recurrent neural networks or convolutional neural networks [2], which has led to significant improvements in tasks such as machine translation [3].

The use of transformer models is not limited to the field of natural language processing (NLP) but has also been expanded to image processing. One prominent implementation is the use of transformer models in image processing, known as Vision Transformer (ViT) [4]. To support the performance of ViT, a dedicated accelerator has been developed called ViT Accelerator (ViA) [5].

The transformer model itself is still in the stage of efficiency and optimization, ranging from accuracy, and speed, to computational flexibility. Research on transformer models using Field Programmable Gate Array (FPGA) has been conducted [6], opening up opportunities for further research. We conducted experiments as follows:

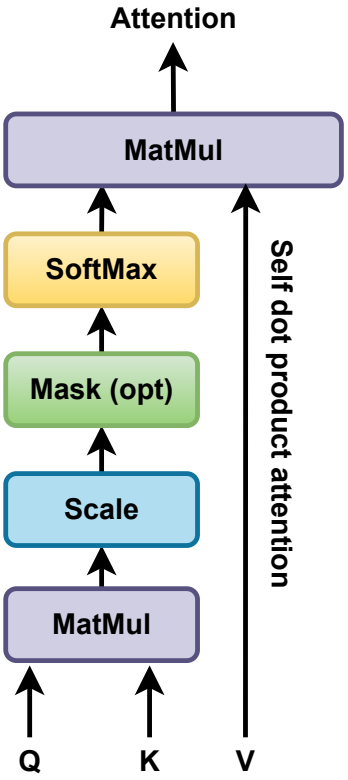

Fig. 1: Self head attention [1]

- Utilizing the systolic array property in matrix multiplication by evaluating and developing the matrix computation process using parallel partitions.
- Proposed architecture to be used for conducting experiments using FPGA Zybo 7020 hardware.

Therefore, the objectives of conducting these experiments are as follows:

- The main objective is to accelerate the transformer model specifically in self and multi-head attention.
- Aims to enhance performance with limited resources by evaluating and developing the matrix computation process.
- By minimizing the available resources, a lightweight model can be created.

The continuation of this paper consists of several chapters, arranged as follows. Chapter II will discuss the theoretical foundations that support the methods to be used, along with

several related works relevant to the proposed method. Chapter III contains the proposed method, including the method and architecture to be proposed. Chapter IV presents how the experiment is conducted. Chapter V contains the experimental results. Chapter VI discusses the conclusion and future development.

II. Related works

A. Self-Head Attention

In broad terms, the transformer model consists of three parts: word embedding, encoder blocks, and decoder blocks. Within the decoder block, there is a component called Attention used to compute the relationships between words simultaneously. Within this, vectors are collected to form matrices of query (Q), key (K), and value (V). These matrices are then computed into Self-Head Attention using the following formula [1]:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

B. Multi-Head Attention

Multi-head attention is the application of several self-head attentions running in parallel with separate parameters. After the calculation of each head is completed, the results of these calculations are combined. This can help the model capture more complex relationships between words and enable the model to represent the input data better, resulting in richer and more contextual outcomes, thus addressing more complex tasks. Multi-head attention is computed according to the following formula [1]:

$$Multihead(Q, K, V) = Concat(head1, ..., head_n) \quad (2)$$

C. Systolic Array

A systolic system is an architecture designed to perform processes in parallel, using multiple simple processing elements. Simplicity with orderly communication and control structures have great advantages over complicated structures in design and implementation, the cells in the systolic system are usually interconnected to form a systolic array or systolic tree.

Since early 1984, CMU has been working on the design of Warp engines using systolic processors, which were intended to be fast and powerful computing engines for many low-level signal processing and image processing tasks [7]. The Warp processor family uses a systolic array that can efficiently implement a variety of systolic algorithms, including Fast Fourier Transform, matrix multiplication, and two-dimensional convolution [8].

Replacing processing elements (PE) with PE arrays, or cells as shown in Fig.3.b can achieve higher computing throughput without increasing memory bandwidth. Each data from the main memory is processed several times and sent back to memory [9].

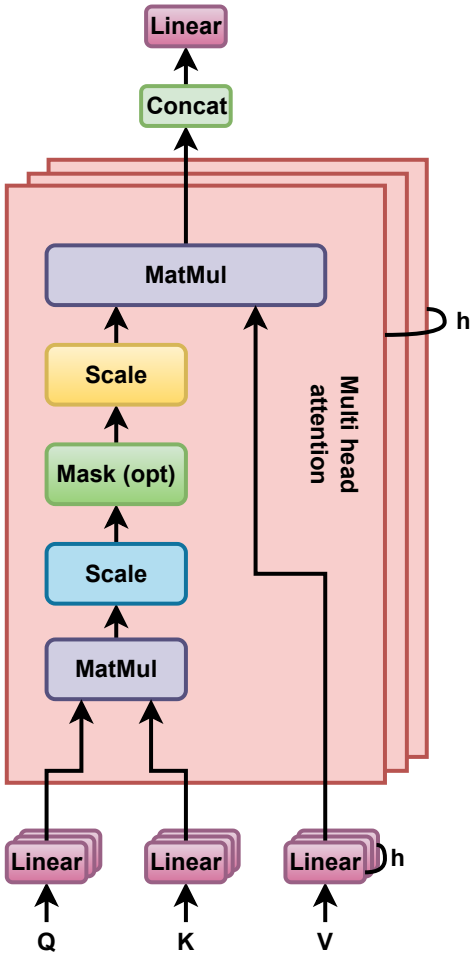

Fig. 2: Multi head attention [1]

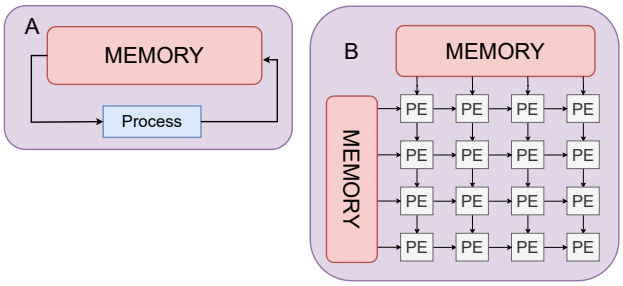

Fig. 3: A) Normal computation. B) Systolic Array

III. Proposed Method

A. Parallel Partition

Parallelization here involves partitioning and paralleling the computation of large-dimensional matrices. Matrix partitioning consists of breaking down large-dimensional matrices into several smaller-dimensional matrix parts. In this case, we use a 4×4 smaller matrix. We use the BERT model as the foundation of our method. The BERT model utilizes a 768-dimensional vector [10]. With the dimensions 4×768, we will partition it into 4×4, which will iterate 192 times.

In Fig.4, a matrix Q with dimensions of 4×768 will be multiplied by a large matrix K^T with dimensions of 4×768.

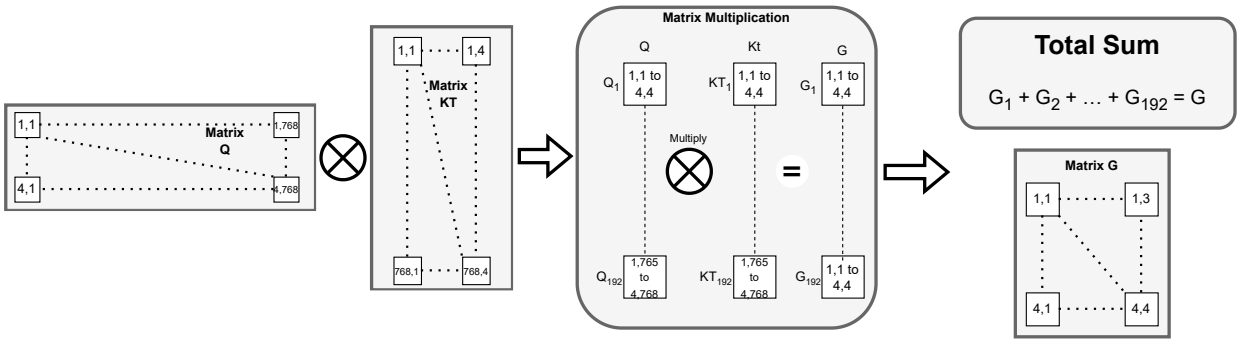

Fig. 4: Q and Kt partition to form matrix G

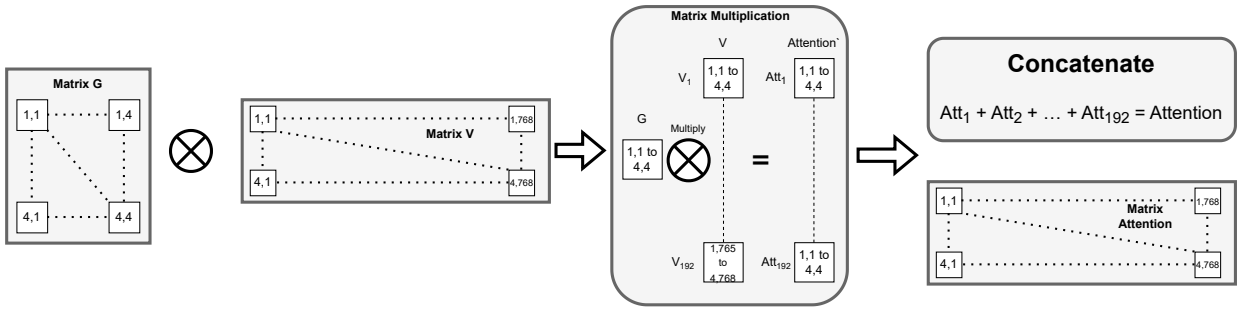

Fig. 5: V partition to form Attention

The matrices Q and K^T will be partitioned into 4×4 small matrices, Q_{1-192} and K^T_{1-192}, which will then be multiplied pairwise to form temporary matrices G_{1-192} with dimensions of 4×4. G_{1-192} will then be summed up entirely to form the matrix G with dimensions of 4×4.

In Fig.5, the matrix V with dimensions of 4×768 will be partitioned into small matrices V_{1-192} with dimensions of 4×4. Each V_{1-192} will be multiplied by the matrix G. The multiplication result is Att_{1-192}, each with dimensions of 4×4. To obtain the final Attention matrix, matrices Att_{1-192} will be sequentially concatenated until the Attention matrix has dimensions of 4×768.

B. Architecture

Fig.6 provides an overview of our hardware setup, utilizing 2 Block RAM (BRAM) modules as input and 1 BRAM module as output. The IP Core contains signal control and a systolic array consisting of multiple Processing Elements (PEs).

Fig.7 is our proposed processing element. The input from matrices A and B is kept in the register and will be forwarded in the next cycle. A is multiplied by B to get output C, and the result will be maintained and accumulated in the next cycle to get the desired output.

IV. Experiment

A. Systolic Array & Bram

We use Vivado 2019.1 in this experiment. We first created the processing element and integrated it with the systolic array. We use algorithm 1 as the processing element. If the $reset$ signal is true, all outputs will equal 0. Else, $Output\_C$ is calculated with $Output\_C + (Input\_A \times Input\_B)$, while $Forward\_A$ and $Forward\_B$ are filled with $Input\_A$ and $Input\_B$ respectively.

We implement finite state machine (FSM) as shown in Fig.8 into the main code. The system will start at Idle state, waiting for the process to begin with a $start$ signal. If the $start$ signal is active and the num_of_input parameters are zero, it moves to the Done state, resetting the $start$ signal, and moving again to the Idle state. Otherwise, it moves to the Read state, where the data from $Bram\_A$ and $Bram\_B$ will be read, and the $write - enabled$ signal activated. At the Write state, data from $Bram\_A$ and $Bram\_B$ that have been processed will be written into $Bram\_C$ as the output. If the data that came in does not equal the num_of_input parameter, it will return to the Read state. If equals, the next phase is in the Done state, resetting the $start$ signals, and moving to Idle states awaiting the next process or cycle.

Algorithm 1 Processing Element

Input : $reset$**,** $clock$**,** $InputA$**,** $InputB$
Output : $Forward\_A$**,** $Forward\_B$**,** $Output\_C$
procedure ALWAYS @(POSEDGE CLOCK)
  if $reset$ IS TRUE **then**
    SET $Forward\_A$, $Forward\_B$, $Output\_C$ TO 0
  else
    SET $Output\_C$ TO $Output\_C + (Input\_A \times Input\_B)$
    SET $Forward\_A$ TO $Input\_A$
    SET $Forward\_B$ TO $Input\_B$
  end if
end procedure

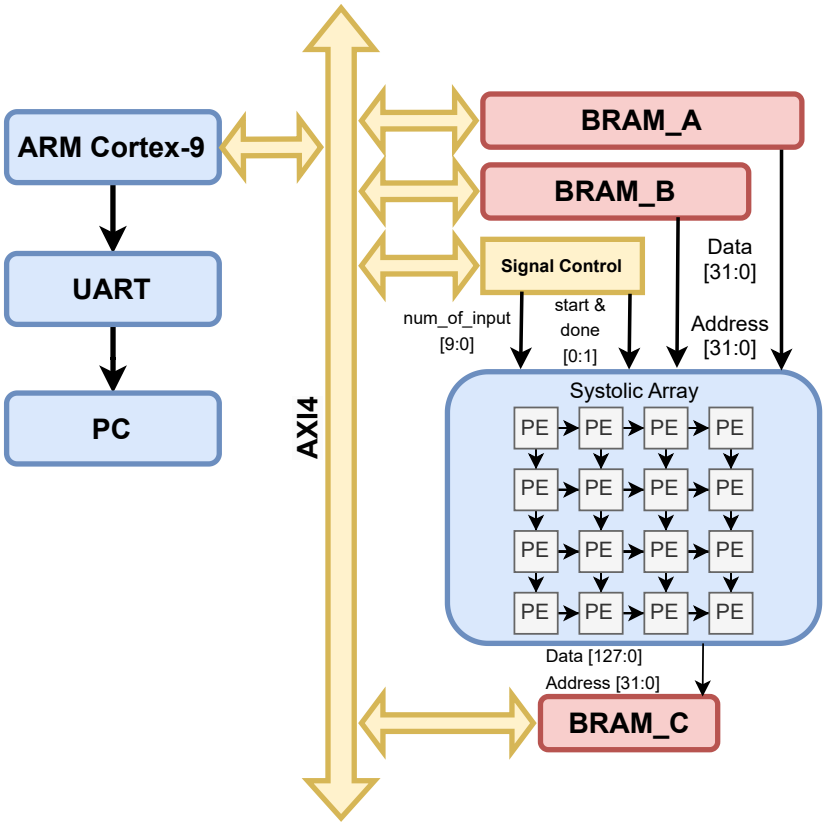

Fig. 6: Proposed Architecture Overview

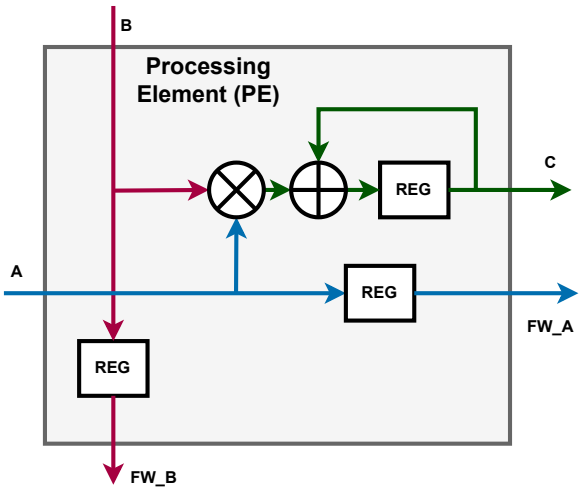

Fig. 7: Architecture of processing element

B. Parallel Partition

We simulate the parallel partition in Python. For the first part, we multiply the matrix Q and K^T. Same as Fig.4 with algorithm 2. The second part is multiplying the matrix result from the first part (matrix G) to matrix V, same as Fig.5 with algorithm 3.

Algorithm 2 $Q \times K^T$ partition

Input : $QMatrix$, $K^T Matrix$
Output : $GMatrix$
SET $block\_size$ TO 4
for i **do** FROM 0 TO 768 STEP 4 DO
SET $block\_Query$ TO $QMatrix$ [ALL ROW, i TO $i+block\_size$]
SET $block\_Key^T$ TO $K^T Matrix$ [i TO $i+block\_size$, ALL COLUMN]
SET $block\_result$ TO DOT PRODUCT OF $block\_Query$, $block\_Key^T$)
ADD $block\_result$ TO $GMatrix$
end for
PRINT $GMatrix$

Algorithm 3 $G \times V$ partition

Input : $GMatrix$, $VMatrix$
Output : $Attention$
SET $block\_size$ TO 4
for s **do** FROM 0 TO 768 STEP 4 DO
SET $block\_Value$ TO $VMatrix$ [ALL ROW, s TO $s+block\_size$]
SET $block\_attention$ TO DOT PRODUCT OF ($GMatrix$, $block\_Value$)
CONCATENATE $block\_attention$ TO $Attention$
end for
PRINT $Attention$

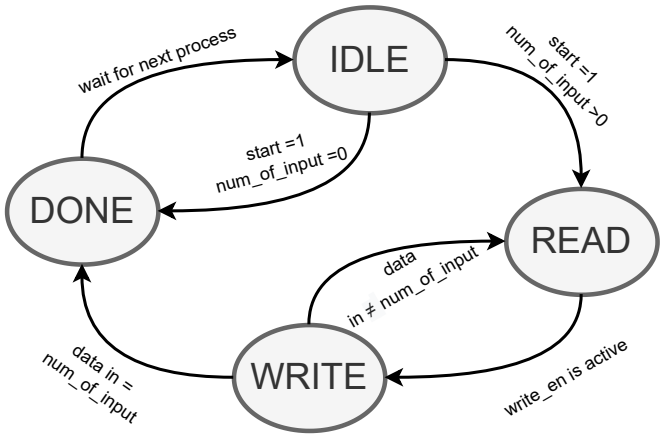

Fig. 8: Finite state machine

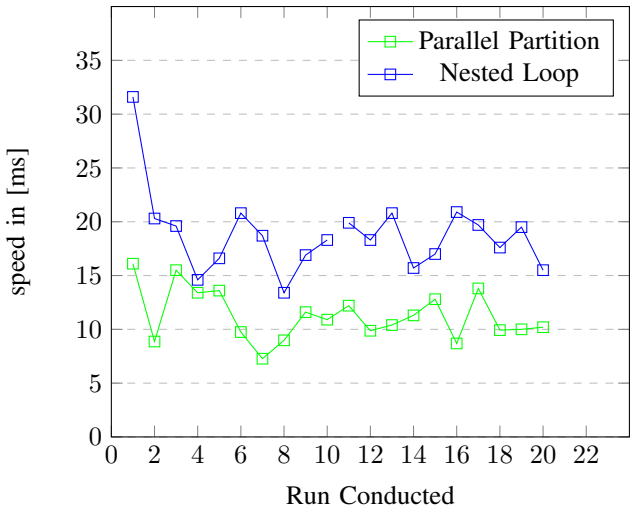

Fig. 9: Python simulation results

V. Experimental Results

Fig.9 is the result of the simulation of algorithm 2 and 3 from Python by using Google Colab. While Fig.10 is using C in CodeBlocks. For comparison, we include matrices multiplied only using a conventional nested loop in Python. From conducting 20 times runs, we get the slowed method is a conventional nested loop with 31.60 ms as the slowest and 13.40 ms as the fastest, averaging at 18.79 ms. Then followed by parallel partition in Python with 16.10 ms as the slowest and 7.27 ms as the fastest, averaging at 11.26 ms. The faster is the parallel partition in C, with 0.28 ms as the slowest and 0.19 ms as the fastest, averaging 0.22 ms.

Using the design in Fig.6 we run synthesis and implementation in Vivado 2019.1 without any problem. The worst negative slack (WNS) result is 9.1 ns with a clock period of 20 ns. For finding max frequencies we use equation 3. We get 0.0917 Ghz or 91.7 Mhz. Table I breaks down the utilization and report timing.

$$F_{max} = 1/(T - WNS) \tag{3}$$

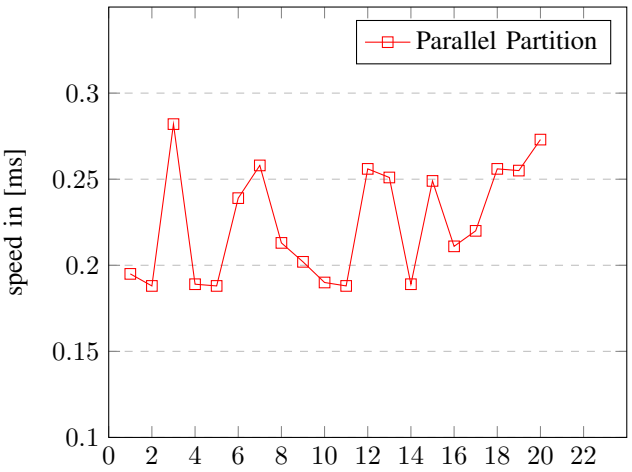

Fig. 10: Parallel Partition in C

Site Type	Used	Fixed	Available	Util%
Slice LUTs	8354	0	53200	15.70
LUT as Logic	6622	0	53200	12.45
LUT as Memory	1732	0	17400	9.95
LUT as Distributed RAM	1308	0		
LUT as Shift Register	424	0		
Slice Registers	9309	0	106400	8.75
Register as Flip Flop	9309	0	106400	8.75
Register as Latch	0	0	106400	0.00
F7 Muxes	6	0	266000	0.02
F8 Muxes	0	0	13300	0.00
Block RAM Tile	8	0	140	5.71
RAMB36/FIFO	8	0	140	5.71
RAMB36E1 only	8			
RAMB18	0	0	280	0.00
DSPs	0	0	220	0.00

TABLE I: Timing and utilization report

VI. Conclusions and Future Works

A. Conclusion

This experiment simulation and hardware test demonstrate that parallel partition is faster than the conventional nested loops in Python. The process speed increase averaged around 66.9%. Calculating parallel partitions with C language proved even faster, with an average speed of 0.22 ms. Additionally, the proposed architecture synthesis and implementation design using Zybo 7020 with Vivado 2019.1 showed maximum frequencies of 91.7 Mhz without issues.

B. Future Works

This experiment finds several upgrades for future works that can be pursued to enhance the performance and capability of the transformer model, such as:

- Hardware implementation: Our future work is implementing the design and architectures into the zybo 7020 hardware.
- Scaling into larger and more complex models of the transformer: Our experiment can be expanded to handle larger data sets. With more layers and parameters, the model

can capture more complicated patterns and dependencies on data. As a result, increasing the accuracy, making the model more reliable.

- The parallel partition can be applied to other machine learning models: Not only the transformer model but to other models using matrix multiplication. By optimizing the models using parallel partitions, we can achieve performance gains.
- Optimizing beyond natural language processing: It can be optimized not only as natural language processing but various domains such as robotics and computer vision. Expanding the application in various scopes.

ACKNOWLEDGEMENT

This work is supported by the 2024 ITB Young Researcher Program provided by LPPM, Bandung Institute of Technology under grant no. [STEI1.PN-6-04-2024].

REFERENCES

[1] A. Vaswani et al., *Attention Is All You Need* arXiv, Aug. 01, 2023. Accessed: Dec. 03, 2023. [Online]. Available: http://arxiv.org/abs/1706.03762

[2] H. Peng, R. Schwartz, D. Li, and N. A. Smith, *A Mixture of $h-1$ Heads is Better than h Heads* arXiv, May 13, 2020. Accessed: Dec. 03, 2023. [Online]. Available: http://arxiv.org/abs/2005.06537

[3] S. M. Lakew, M. Cettolo, and M. Federico, *A Comparison of Transformer and Recurrent Neural Networks on Multilingual Neural Machine Translation*. *In Proceedings of the 27th International Conference on Computational Linguistics*, New Mexico, USA: Aug.2018 pp. 641–652.

[4] Vaswani, Ashish, et al. *A*ttention is all you need. Advances in neural information processing systems 30 (2017).

[5] Dosovitskiy, Alexey et al. *A*n Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. ArXiv abs/2010.11929 (2020): n. pag.

[6] B. Li et al., *FTRANS: Energy-Efficient Acceleration of Transformers using FPGA* vol. 7, 2020, doi: https://doi.org/10.1145/3370748.3406567.

[7] Kung, H.T. *Systolic Algorithms for the CMU Warp Processor*, *In Proceedings of the Seventh International Conference on Pattern Recognition*, pages 570-577.1984.

[8] E. Arnould, H. Kung, O. Menzilcioglu and K. Sarocky, *A systolic array computer*, ICASSP '85. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Tampa, FL, USA, 1985, pp. 232-235, doi: 10.1109/ICASSP.1985.1168488.

[9] Kung, *Why systolic architectures?*, in *Computer, vol. 15*, no. 1, pp. 37-46, Jan. 1982, doi: 10.1109/MC.1982.1653825.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding arXiv*, May 24, 2019. Available: http://arxiv.org/abs/1810.04805