Performance Analysis of Parallelization on Vision Transformer

Mustafa İzzet Muştu
Department of Computer Engineering
İstanbul Technical University
İstanbul, Turkey
mustu18@itu.edu.tr

***Abstract*—In recent years, Vision Transformer (ViT) has emerged as a powerful model for image recognition tasks, achieving state-of-the-art performance on several benchmarks. However, the large size of ViT models and the complexity of their attention mechanism make them computationally expensive, limiting their scalability and applicability to real-world problems. To overcome this challenge, parallelization on GPUs has been proposed as an effective solution. In this paper, we investigate the parallelization of ViT on GPUs to speed up evaluation process and improve the efficiency of the model. We explore different parallelization techniques that introduce a hybrid approach that combines data and model parallelism to minimize the communication overhead and optimize the performance of the system.**

***Index Terms*—computer vision, parallelizing, CUDA**

I. Introduction

Deep learning models, especially convolutional neural networks (CNNs), have become the de facto standard for image recognition tasks. However, recent studies have shown that the attention mechanism used in the Transformer model can achieve comparable or better performance than CNNs, especially for long-range dependencies and complex visual patterns. Vision Transformer (ViT) [1] is a notable example of a Transformer-based model that has shown promising results in image recognition.

Despite its performance, ViT has one major limitation. It's size and high computational requirements make it impractical for many real-world scenarios. To address this challenge, parallelization methods have been proposed to speed up training and improve efficiency. An effective approach is to parallelize ViT on GPUs. This significantly reduces training time and improves scalability.

This paper focuses on parallelizing ViT as well as input data on GPUs to analyze the parallelization effect on the inference time of the model. Several different configurations have been experimented and results are reported.

The rest of this paper is organized as follows: Section 2 provides background information on ViT and parallelization techniques, and Section 3 explains the parallelization implementation. Section 4 describes the experimental setup and results, and Section 5 presents the discussion and conclusions. Finally, Section 6 suggests what can be done in the future to increase performance.

II. Related Work

A. Attention and Multi-Head Self Attention

A process that generates an output from a query (Q) and a set of key-value (K-V) pairs is known as an attention function. These are all expressed as vectors, including the queries, keys, values, and output. The result is calculated by adding the values in a weighted total, where the weight given to each value is established by a compatibility function that assesses how comparable or compatible the query and the associated key are. In practical applications, the attention function is computed for a group of queries simultaneously, where the queries are combined into a matrix Q. Similarly, the keys and values are packed together into matrices K and V, respectively. This allows for efficient computation and parallel processing of the attention mechanism, enabling faster and more scalable operations on the input data. A commonly preferred version of this mechanism that is also used in ViT is Scaled Dot-Product Attention. It is calculated as shown in Equation 1, where d_k is the dimension of the query, key and value.

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

It is useful to apply linear projections to the queries, keys and values several times to improve the performance and efficiency of the attention mechanism. In [2], these linear projections use learned linear transformations to translate the queries, keys, and values into distinct dimensions, namely d_k for keys and queries, and d_v for values. Then, output values with dimensions of d_v are produced by applying the attention function in parallel to each of these projected versions. The final values are created by concatenating these obtained values and further projecting them. In [2], it is mentioned that this multi-head self attention (MSA) method improves the attention mechanism's flexibility and information extraction. Calculation of MSA can be seen in Equation 2, where h is the number of heads. Both single and multi head self attention mechanisms can be seen in Fig. 1.

$$MSA(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{2}$$

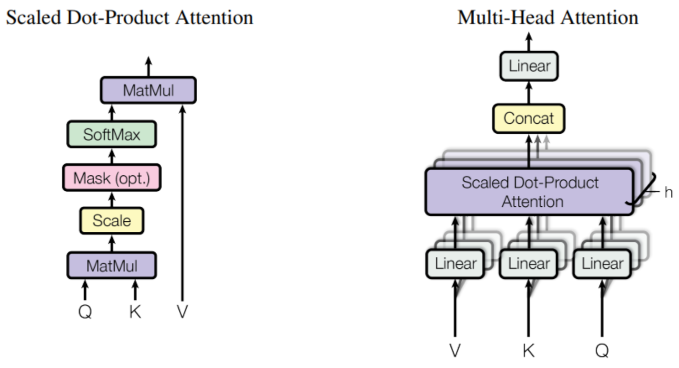

Fig. 1. Demonstration of scaled-dot product attention and multi-head self attention [2]

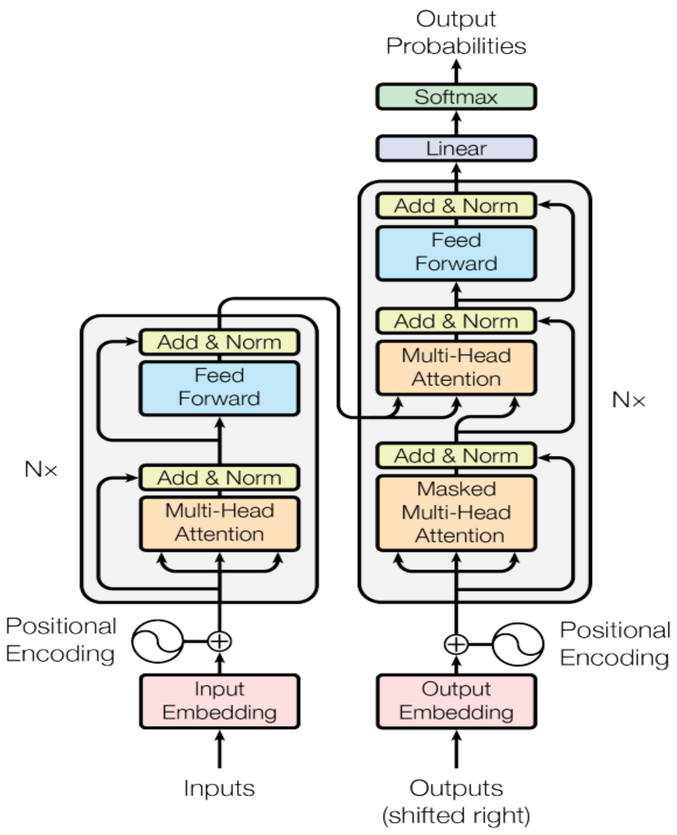

Fig. 2. Transformer architecture [2].

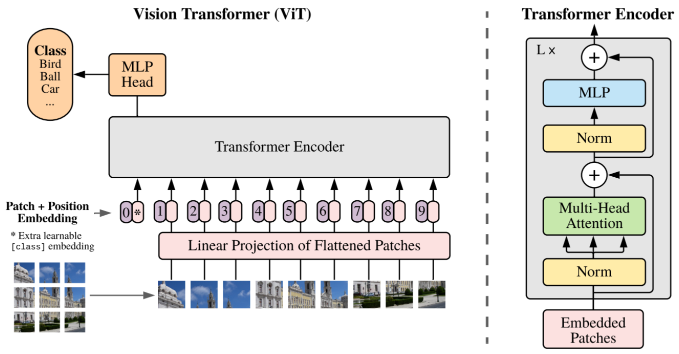

Fig. 3. Vision Transformer (ViT) architecture [1].

B. Transformer

The transformer [2] model consists of two main components: the encoder and the decoder. The encoder processes the input sequence, while the decoder generates the output sequence. Both the encoder and decoder comprise multiple layers of self-attention mechanisms and feed-forward neural networks. The self-attention mechanism plays a crucial role in capturing relationships among the words or tokens in the input sequence. By calculating attention weights, it determines the significance of each word relative to others in the sequence. This mechanism enables the model to focus on relevant words and effectively handle long-range dependencies. To preserve the positional information of words, the transformer model introduces positional encoding. The architecture of the Transformer model is shown in Fig. 2.

C. Vision Transformer

The transformer architecture, which was initially created for natural language processing, is used by the Vision Transformer (ViT) model to analyze picture data. ViT divides the input image into patches of a predetermined size and arranges them in a grid-like pattern. Each patch is handled as a token and put through a linear embedding procedure to create a representation in a lower-dimensional vector space. Positional embeddings are combined with patch embeddings to encrypt spatial information.

The series of patch embeddings is then run through a typical transformer encoder, which is made up of numerous layers of feed-forward neural networks and self-attention processes. Thorough knowledge of the complete image is made possible by the self-attention mechanism, which enables the model to recognize connections and dependencies between the patches. The information is subsequently processed by the feed-forward networks to identify significant characteristics and patterns. The finished image representation is achieved when the transformer encoder with L number of layers has been applied. To forecast the class label for the image, this representation can be passed further into a classification head, such as a fully connected layer. The architecture of the model can be seen in Fig. 3.

D. Parallelizing Deep Learning Models

To increase the effectiveness of training deep learning models, parallelization includes distributing computational activities across several processing units, such as GPUs or numerous machines. Deep learning models can be parallelized using a variety of techniques, each of which has advantages and things to keep in mind.

Data parallelism [3] is a widely utilized technique that involves distributing the training data among various processing units. Each component separately analyzes a portion of the data and calculates gradients. The parameters of the model are then updated by averaging or combining these gradients. By splitting the data into multiple smaller batches, data parallelism improves resource use, allows for training on larger datasets, and aids in overcoming memory constraints.

Model parallelism [4], [5], [6] is a different strategy that focuses on dividing the model's parameters among many processing units. The model's various portions are assigned to various units, and computations are run simultaneously in each unit. When working with really huge models that won't fit in the memory of a single device, model parallelism is especially helpful. The model can be divided, allowing computations to be carried out in parallel and cutting down on training time.

A complete solution for training massively parallel deep learning models is provided by hybrid parallelism [7], which includes data parallelism and model parallelism. Utilizing the benefits of each strategy entails parallelizing the data and the model. For large models that call for memory and computation efficiency improvement, hybrid parallelism is very beneficial.

III. Methodology

For a fair comparison between the CPU and GPU run models, the exact same ViT-Base model without dropout from the original paper is implemented from scratch by using C++ with Eigen [8] for CPU and CUDA with CUBLAS [9].

A. Parallelization on CPU

The most noticeable fact from the ViT architecture is that it consists of layers that accept input from the output of the previous layer. When we observe the one layer of the encoder, we can easily distribute the job on the multi-head self attention part with multiple threads. Other than that, we can adopt multi-threaded matrix multiplication for each matrix multiplication and we can simultaneously calculate the softmax operation for each row in the provided matrix. Experimental results are shared in Section 4.

B. Parallelization on GPU

Parallelization on GPU

IV. Experiments

Experiments are conducted on a system that has Windows 11 22H2 operating system and nvcc version V12.1.66. Hardware information is as follows: AMD Ryzen 7 5800X, 32 GB 3400 MHz, NVIDIA RTX 3060 12 GB.

CPU execution times of all layers in the ViT-Base encoder are measured. Patch embeddings, layer normalization in the last layer and averaged results for repeating parts can be seen in I. It is worth mentioning that although each head in MSA is separated into different threads, most of the time is taken by this part of the encoder. This is caused by the attention mechanism that requires high-degree matrix-matrix multiplications. CPU thread usages are reported as %100 for this part.

V. Conclusion

Conclusion

TABLE I
Execution times in 1 layer of the ViT-Base encoder (CPU).

Operation	**Time (s)**
Embeddings (Only before the first layer)	1.190
Layer Normalization	0.024
MSA (Multithreaded)	21.220
Residual	0.001
Layer Normalization	0.024
MLP	9.602
Residual	0.002
Layer Normalization (Only after the last layer)	0.0231

VI. Future Work

Future Work

References

[1] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[3] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl, "Measuring the effects of data parallelism on neural network training," 2019.

[4] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," 2020.

[5] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[6] C. Karakus, R. Huilgol, F. Wu, A. Subramanian, C. Daniel, D. Cavdar, T. Xu, H. Chen, A. Rahnama, and L. Quintela, "Amazon sagemaker model parallelism: A general and flexible framework for large model training," 2021.

[7] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," 2022.

[8] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[9] "Cublas," https://developer.nvidia.com/cublas/, accessed: 2023-05-12.