Performance modeling 3D Vision Transformers

Varuni Sastry, Eugene Ku

March 11, 2025

Abstract

A 3D Vision Transformer (3D ViT) is an extension of the Vision Transformer (ViT) architecture designed to process 3D data, such as videos, and volumetric images (e.g. medical scans like CT or MRI). These kind of vision models are specifically important to ALCF users targeting APS volumetric data such as processing BCDI data or spatio-temporal inputs from climate applications. Specifically, we stress-test the backbone model of VIT which also generalizes to newer VIT variants. We use a transformer based no-mask encoder to processes the sequence of 3D patch embeddings using self-attention mechanisms and feed-forward layers. For the first set of experiments we will use MHA, and explore sparse attention in our future works. We a linear layer to model the classification task specific decoder, which can be swapped out depending on the downstream task at hand. Additionally, we emphasize high data volume (sequence length); thus leading a system that can not only tolerate heavy IO but also frequent and massive communication patterns of parallelisms for sharding a single data sample like with sequence parallelism (Ulysses-based implementation).

1 Terms and definition

Below is the outline for the 3D vision transformer model.

- **Input sample**: The input is of shape (b_s,C,D,H,W), where b_s=batch, C=number of channels, D=depth, H=height, W=width of the 3D data.
- **Sequence length (s)**: Each sample in the training is an image of size (D,H,W) as explained above. Each of them is patched into small chucks that are called tokens. The number of tokens in a sample is called sequence length.
- **Patched embedding dimension (d)**: each token is represented as a vector in a high dimensional space. The dimension of that space is called the embedding dimension.
- **Input (X)**: Each input token is embedded into a vector of d dimension and the input sample to the transformer block is a is a 2-D tensor of $[s, d]$.
- **Number of attention head (h_c)**: this is the number of Q, K, V matrices in the multi-head attention; the dimension of each matrix is $[d, d_k]$ where $d_k = d/h_c$.

- **Positional Encodings/Embedding**: Positional encodings/embeddings are added to the patch embeddings to retain information about the order and location of patches. Instead of learned function (positional embedding), we use 1D fixed sinusoidal function (positional encoding) to prevent explosion of the number of parameters when sequence length is high.
- **Sequence parallelism size** SP: Number of ranks in the sequence parallel group.
- **Data parallelism size** DP: Number of ranks in the data parallel group.

2 Forward propagation

In each of the transformer layer, there are two parts (1) self-attention layer and (2) multilayer perceptrons (MLP).

2.1 Self-attention

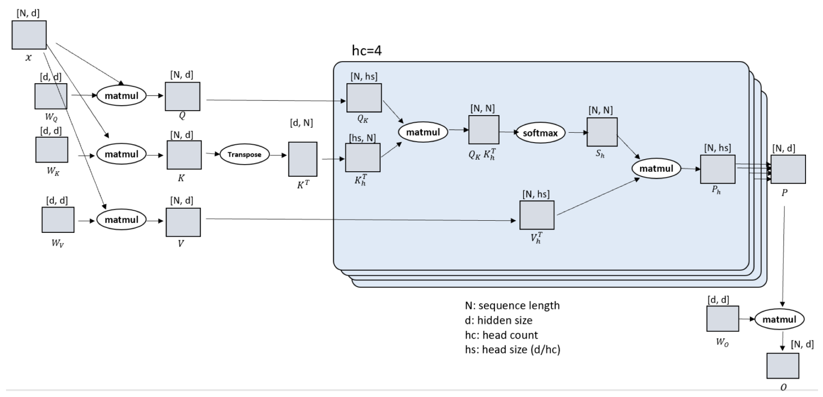

Figure 1: Self-attention layer

We assume the input matrix is X of $s \times d$ where s is the sequence length and d is the hidden dimension. In the self-attention layer, we have

$$Y = \text{softmax}(QK^T)V\,, \tag{1}$$

where Q, K, V are the query, key, and value matrices,

$$V = XW_V, \quad Q = XW_Q, \quad K = XW_K\,. \tag{2}$$

With only DP in place, the Query, Key, Value projections is equivalent to

$$[s, d] \times [d, d] \equiv [s, d] \tag{3}$$

And the total number of FLOPs associated with these GEMM operations are

$$\#FLOP = 3 \times s \times d \times d = 3sd^2 \tag{4}$$

In multi-head attention, instead of one single Q, K, V matrix of dimension $[d, d]$, we have h sets of matrices Q_i, K_i, and V_i, each is of dimension $s \times d_k$, where $d_k = d/h$, which is the result of XW_M, for $M = Q, K, V$.

$$[s, d] \times [d, d_k] \equiv [s, d_k] \tag{5}$$

With Sequence parallelism enabled, each GPU as shown in figure has a subset of s tokens but with full embedded dimension $d = h_c \times d_k$, where h_c is the total number of head counts.

$$[\frac{s}{SP}, d] \times [d, d] \equiv [\frac{s}{SP}, d] \tag{6}$$

$$\#FLOP = 3 \times s/SP \times d \times d = \frac{3sd^2}{SP} \tag{7}$$

Note that in computing Q, K, and V, different subset of tokens are distributed to different SP ranks, each owns s/SP tokens but with full embedded dimension d for each token.

Before computing softmax of the attention, we need to gather the entire sequence to compute full attention but now distributed over the heads. So with all2all communication, the Q, K, and V matrices that were of size

$$[\frac{s}{SP}, d] \tag{8}$$

is transformed to

$$[s, d/h_c \times h_c/SP] = [s, d/SP] \tag{9}$$

The GEMM operation for $Q.K^T$ equals,

$$[s, d/SP] \times [d/SP, s] = [s, s] \tag{10}$$

and the total flop count per rank equals

$$s \times d/SP \times s = \frac{s^2 d}{SP} \tag{11}$$

For computing the softmax on top of the $Q.K^T$, flops equals s^2. And when multiplying with vector V, equals to

$$[s \times s] \times [s, d/SP] = [s, d/SP] \tag{12}$$

with flops equal to

$$s \times s \times d/SP = \frac{s^2 d}{SP} \tag{13}$$

There is another all2all that is required now to re-distribute the input tensor split on the sequence dimension rather than the head dimension. Now the input tensor is $[s/SP \times d]$ and is passed through a fully connected output layer.

$$[s/SP, d] \times [d, d] = [s/SP, d] \tag{14}$$

stage	operation	FLOP	matrix
Generating Q, K, V	gemm	$3sd^2/SP$	$[s/SP, d] \times [d, d]$
$Q_iK_i^T$	gemm	s^2d/SP	$[s, s] \times [s, d/SP]$
softmax	exp	s^2	[s, s]
softmax $\times V$	gemm	s^2d/TP	$[s, s] \times [s, d_k]$
FCL	gemm	sd^2/SP	$[s/SP, d] \times [d, d]$
SA	all2all	0	tbd

Table 1: Operations involved in self-attention

and will account to the following total flops.

$$s/SP \times d \times d = \frac{sd^2}{SP} \tag{15}$$

Now let us summarize the computation / communication involved in the self-attention Total number of flops is $4sd^2/SP + 2s^2d/SP + s^2$

The output from self-attention layer will be passed to the multilayer perceptrons. The input dimension is $[s/SP, d]$, where s is the sequence length and d is the hidden dimension. The shapes of two MLP layers are $MLP1 \equiv [d, 4d]$, and $MLP2 \equiv [4d, d]$ respectively. MLP1 layer account to the following GEMM operation,

$$[s/SP, d] \times [d, 4d] = [s/SP, 4d], \tag{16}$$

which amounts to the following flops,

$$s/SP \times d \times 4d = 4sd^2/SP, \tag{17}$$

MLP2 layer account to the following GEMM operation,

$$[s/SP, 4d] \times [4d, d] = [s/SP, d], \tag{18}$$

which amounts to the following flops,

$$s/SP \times 4d \times d = 4sd^2/SP, \tag{19}$$

Therefore the total flops count for the MLP layer equals $8sd^2/SP$.

The total forward pass FLOPs equals,

$$(\frac{4sd^2}{SP} + \frac{2s^2d}{SP} + s^2) + \frac{8sd^2}{SP} = \frac{12sd^2 + 2sd^2}{SP} \tag{20}$$

3 Backward propagation

With the rule of thumb, the back propagation has about two times FLOPs of the forward.

4 Memory Footprint

The model parameters W_K, W_Q, W_V, W_o from the self-attention layer, are of tensor shape $[d, d]$; MLP layer has two weight matrices $[4, 4d]$, and $[4d, d]$. Hence, the total number of parameters in a transformer layer $\psi = 12Ld^2$ where L is the number of layers in the model.

The total memory footprint for parameter ($\mathbb{P}$), gradients($\mathbb{G}$), and optimizer state ($\mathbb{O}$) are as follows. $\mathbb{P}$ and $\mathbb{G}$ have ψ parameters, while for the optimization process, we have to store the gradient and the momentum. In mixed precision training, we need 6 bytes for each model parameter, 4 to save the model in fp32, and 2 to use in computation in fp16. We need 4 bytes per parameter for Optimizer states to save the momentum in fp32 (Adam Optimizer). We need to save one fp32 gradient value for each parameter. Thus we condider OS to have 6ψ parameters.

With SP, since all the parameters are stored in each rank, and the split happens on sequence dimension, the SP parameter memory footprint is similar to the DP case.

For large models, we use ZeRO3 optimization that will reduce memory by total number device number N_d. In this particular case, the total memory for parameters, gradients and OS are all reduced by a factor of N_d.

One also needs to store the activations, this will be proportional to the batch size. All inputs and activations are listed below:

- Input Embedding X: $(b_s \times s/SP \times d)$
- Q, K, V in self-attention : $(3 \times b_s \times s/SP \times d)$
- $A = \text{softmax}(Q.K^T)$: $(2 \times b_s \times d/SP \times s^2)$
- $B = A.V$ in Attention : $(b_s \times s \times d/SP)$
- $C = B.W_0$ in FCL of self attention : $(b_s \times s/SP \times d)$
- Total SA Activation memory $= (6b_s sd + 2b_s s^2 d)/SP$
- MLP1 $= (4 \times b_s \times s/SP \times d)$
- MLP2 (which is input to next layer) $= (b_s \times s/SP \times d)$

Notice that the total number of parameters are

$$L \times b_s \times (6sd + 2s^2 d + 4sd)/SP\,, \tag{21}$$

where b_s is the microbatch size, and L is the number of layers. Note that we have avoided the double counting of last MLP output as it will be the X in the next layer of self-attention.

5 Communication Complexity

Here we consider the communication complexity of DP, SP and ZeRO3 strategy.

5.1 Data Parallel

For data parallel framework, the only communication for a compute step occurs at the end of the backward pass for the gradient synchronization through an all-reduce call. The total communication volume is equal to the total number of gradients, which is equal to ψ, which is $12\mathcal{O}(d^2)$ [FZ24].

5.2 Sequence Parallel-Ullyses

For sequence parallel framework the gradient synchronization at the end of backpropogation is similar to the DP case. However, additionally sequence parallelism has a total of 8 all2all communication calls, 4 for each forward and backward pass. Of the 4 calls, the 3 all2all communication is for the Q, K, and V vectors, to combine the sequence dimension and split the head dimension just before the actual self-attention. The fourth all2all is to return to the split in the sequence dimension before passing through the fully connected layer W_o. The size of all2all messages that are sent from one rank to another is of size $[s/SP, d/SP]$. So, the total communication cost of the activation parameters is equal to $8/SP\mathcal{O}(b_s sd)$ [FZ24].

5.3 ZeRO3

For the ZeRO3 memory optimization, there are a total of three communication calls, two All-gather calls, one on the parameter weights in the forward pass and the second on the backward pass since the parameters are also distributed locally in ZeRO3. There is another reduce-scatter over the gradients which cumulates to 3ψ. The total communication volume is around $1.5x$ compared to SP parameter/gradient communication cost. The total number of activation communications is 0 for ZerO3 [Raj+20].

The table below summarizes the communication and memory cost for the above three frameworks.

Table 2: Comparison of Communications and Memory Cost of SP, DP, ZeRO3 for a standard transformer block

	Communication (FWD+BWD)				**Split**	**Memory**		
	Param	**Cost**	**Act**	**Cost**	**Dim**	$\mathbb{P}/\mathbb{G}$	$\mathbb{O}$	**Act**
SP-Ulysses	allreduce	$12O(d^2)$	8*all2all	$\frac{8}{N}O(b_s * s * d)$	h_c/s	$\mathbb{P}$+$\mathbb{G}$	$6\mathbb{P}$	A/N
DP	allreduce	$12O(d^2)$	0	0	b_s/b_s	$\mathbb{P}$+$\mathbb{G}$	$6\mathbb{P}$	A/N
ZeRO3	2*allgather+ reducescatter	$18O(d^2)$	0	0	h_c/s	$(\mathbb{P}+\mathbb{G})/N$	$6\mathbb{P}/N$	A/N

6 Definition of the figure of Merit

6.1 Time to Solution

Adding up the total complexity of the forward pass [20], and backward pass (approximating to twice the forward pass complexity), we get the total compute complexity

$$C_{compute} = \alpha b_s \frac{L(36sd^2 + 6s^2d)}{SP} \tag{22}$$

The total memory complexity is given by

$$C_{mem} = b_s L \frac{(10sd + 2s^2d)}{SP} + 12Ld^2 \tag{23}$$

The total communication complexity is given by

$$C_{communication} = L(18d^2 + 8/SP * (b_s sd)) \tag{24}$$

6.2 Figure of Merit

The FOM can be defined as the ratio between the weighted complexities defined in the above section and the time to solution (T). In other words, the compute efficiency per byte of memory is a good metric to evaluate the system performance for the given application.

$$\text{FOM} = \frac{\alpha C_{compute} + \beta C_{communication}}{T.\gamma C_{memory}} \tag{25}$$

But the memory complexity is included in the b_s and the communication complexity is included in the time to solution, and for vision applications, $s \gg d$, where

$$s = \frac{H.D.W}{p^3} \tag{26}$$

So the FOM can be simplified to

$$\text{FOM} = \frac{b_s L(H.D.W/p^3)^2 d}{T}, \tag{27}$$

References

[Raj+20] Samyam Rajbhandari et al. *ZeRO: Memory Optimizations Toward Training Trillion Parameter Models*. 2020. arXiv: 1910.02054 [cs.LG]. URL: https://arxiv.org/abs/1910.02054.

[FZ24] Jiarui Fang and Shangchun Zhao. *USP: A Unified Sequence Parallelism Approach for Long Context Generative AI*. 2024. arXiv: 2405.07719 [cs.LG]. URL: https://arxiv.org/abs/2405.07719.