https://doi.org/10.2352/EI.2024.36.12.HPCI-199

Efficient Distributed Sequence Parallelism for Transformer-based Image Segmentation

Isaac Lyngaas[1†], *Murali Gopalakrishnan Meena*[1], *Evan Calabrese*[3], *Mohamed Wahib*[4], *Peng Chen*[5], *Jun Igarashi*[4], *Yuankai Huo*[6], *and Xiao Wang*[2];
[1] *National Center for Computational Sciences,* [2] *Computational Sciences and Engineering Division, Oak Ridge National Laboratory, Oak Ridge, TN, USA*
[3] *Department of Radiology, Duke University, Durham, NC, USA*
[4] *Riken Center for Computational Science, Tokyo, Japan*
[5] *National Institute of Advanced Industrial Science and Technology, Tokyo, Japan*
[6] *Department of Computer Science, Vanderbilt University, Nashville, TN, USA*
[†] *Email address for correspondence:* lyngaasir@ornl.gov

Abstract

We introduce an efficient distributed sequence parallel approach for training transformer-based deep learning image segmentation models. The neural network models are comprised of a combination of a Vision Transformer encoder with a convolutional decoder to provide image segmentation mappings. The utility of the distributed sequence parallel approach is especially useful in cases where the tokenized embedding representation of image data are too large to fit into standard computing hardware memory. To demonstrate the performance and characteristics of our models trained in sequence parallel fashion compared to standard models, we evaluate our approach using a 3D MRI brain tumor segmentation dataset. We show that training with a sequence parallel approach can match standard sequential model training in terms of convergence. Furthermore, we show that our sequence parallel approach has the capability to support training of models that would not be possible on standard computing resources.

Introduction

Image segmentation is the process of labeling pixels in an image in a manner such that pixels with the same label share certain characteristics, such as belonging to the same structure or region. Computer vision tasks, such as segmentation, have been largely dominated in the past decade by convolutional neural networks (CNNs) due to their powerful feature extraction capabilities. In the realm of image segmentation, one of the most effective networks built around convolutional operators is the U-Net [11]. In fact, many of the top performing networks for various medical image segmentation tasks revolve around adapting the U-Net architecture along with specialized data processing techniques to improve model accuracy corresponding to the specific data that is being trained on. However, the limited kernel size of CNN-based techniques restricts their capability of learning long-range dependencies that are critical for accurate segmentation of features that exist at large context sizes.

Recently, transformer models, which are heavily used in the natural language processing (NLP) realm, have been adapted for the use in the imaging realm via the Vision Transformer (ViT) architecture [4]. There has been a lot of interest to incorporate ViT architectures with imaging based tasks due to the capability to overcome the inherent shortfall in CNN based models, i.e. not being able to capture long range interactions. The major difference between Transformers and CNNs is the use of attention-based operations for modeling long-range information by pairwise interaction between token embeddings rather than convolutional kernels.

A major issue that exists when considering the use of Transformer architectures for imaging data is the inability to efficiently process the underlying attention mechanism for long sequence data. For this reason, ViT models typically implement a patching strategy that creates tokens using neighboring pixels to decrease the sequence length of tokens used as input. This patching strategy is instituted because the memory footprint of transformer models increase quadratically as sequence length grows. Due to limitations in hardware memory, ViT models are often restricted in the size of input and patch size allowed. This limitation prevents the ability to test the performance of models using long sequence inputs.

In order to work around these limitations, various schemes have been devised to approximate these attention operations either through sparse sampling[1, 2, 7, 12, 14] or low rank approximation[3, 6]. However, these schemes suffer from information loss, and it is difficult to assess the extent to which information loss affects model accuracy without proper approaches to efficiently train ViTs with exact attention.

Due to these needs for a proper framework for calculating attention on long sequence data, the authors have been involved in efforts to efficiently implement a technique called sequence parallelism [13]. Sequence parallelism is a memory-efficient parallelism method for transformer models to help break input sequence length limitation and train with longer sequences on GPUs[9]. Our efforts have led to do the development of a techniques that allows for the efficient computation of the attention mechanism for extremely long sequence data. In this work, we expand on these efforts to incorporate a sequence parallel ViT with a down stream task of image segmentation. The novelty within this work is encapsulated in the different approaches used to incorporate the sequence parallel transformer with a downstream image segmentation task.

Method

We are interested in training image segmentation neural networks that are comprised of a ViT encoder equipped with a de-

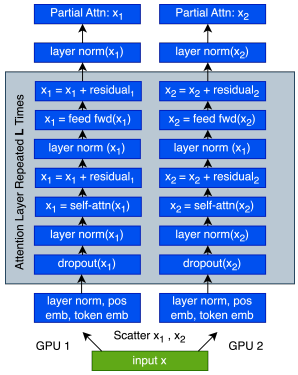

Figure 1. *A generic ViT using a sequence distributed across 2 GPUs with L attention layers where the Attention Layer is outlined by a gray box. Residual$_{1,2}$ represent the input for residual connections, i.e. the input from before being transformed by the self-attention and feed forward layers respectively.*

coder to produce segmentation mapping. In the following sections we detail a Sequence Parallel ViT Encoder, a convolutional decoder implementation for image segmentation, and our strategies for incorporating the two together. These formulations are specifically showcased for a three-dimensional (3D) case to demonstrate the necessary implementation details used in our experimental results.

Sequence Parallel ViT Encoder

Figure 1 illustrates the basic components of a Sequence Parallel ViT encoder. In Figure 1, we show a SeqPar $= 2$ implementation; the expansion to more sequence parallel ranks is evident. The input image x is first distributed in a non-overlapping fashion, namely x_1 and x_2, across two GPUs. A token embedding converts the images into a tokenized image to be used as an input vector[1]. The input vector is then enhanced with embedded positional information. Next, the input vectors of size $\frac{l_x}{2} \times E_m$. Here l_x represents the sequence length and E_m being the embedding size. The input undergoes layer normalization and random dropouts before being processed by a self-attention layer and a feed forward layer with a nonlinear activation function, both equipped with a residual skip connection. On completion, the output consists of what we call a **partial self-attention** output for each input vector.

The essential component of a sequence parallel Transformer lies within how the self-attention computation is handled. Typically, to compute self-attention in a sequential manner, the full input vector x is linearly transformed into query (Q), key (K), and value (V) vectors, with all three vectors having the same size as input x. Then self-attention computes the following:

$$E = \text{softmax}\left(QK^T/\sqrt{d_k}\right)V = A_w V \ . \tag{1}$$

Here, the self-attention score, denoted as A_w, has size $l_x \times l_x$ and quantifies the correlation between each pair of tokens. This correlation is calculated by taking the dot product between Q and the transposed K. To stabilize gradients during training, the self-attention score, A_w is further scaled by a constant $\sqrt{d_k}$, and is then normalized by a dropout and a SoftMax activation. The final self-attention output E is obtained by weighted averaging between value vector V and A_w.

With sequence parallelism, we parallelize the self-attention calculation. This is done by distributing the query vector, Q, and computing the self-attention output as the following concatenation in the sequence dimension,

$$E = \begin{bmatrix} \text{softmax}\left(Q_1K^T/\sqrt{d_k}\right)V \\ \text{softmax}\left(Q_2K^T/\sqrt{d_k}\right)V \\ \text{softmax}\left(Q_3K^T/\sqrt{d_k}\right)V \\ \vdots \end{bmatrix} , \tag{2}$$

where Q_i is an $\frac{l_x}{N} \times E_m$ distributed query vector segment received by the i^{th} GPU, with N being the total number of GPUs. V and K are $l_x \times E_m$ collected value and key vectors, having the same copy across all GPUs. $\text{softmax}(Q_iK^T/\sqrt{d_k})V$ represents the i^{th} GPU's self-attention output for its assigned segment, giving the partial self-attention in the context of the whole sequence. However, in practice, it was deemed unnecessary to perform this concatenation. Instead, it was found that loss calculations can simply be performed on the separate partial self-attention outputs and aggregated as long as the model weights are updated by averaging gradients across the sequence parallel GPUs. For more details on this process, we refer the readers to [13].

The sequence parallel implementation of [13] is able to efficiently perform self-attention computations for a sequence parallel Transformer model with minimal computations and communications. The key takeaway from this sequence parallel ViT encoder implementation is that the output consists of separate partial self-attention outputs on each sequence parallel rank. In this work we investigate the proper approach to use these sequence parallel transformer outputs on a downstream task of image segmentation.

Convolutional Decoder

A common choice of neural network that is especially prevalent for tasks such as image segmentation is the encoder-decoder style architecture. An encoder-decoder architecture will first take input data and encode features into a latent space. These latent space features are then passed through a decoding stage to generate a prediction.

Given the success of encoder-decoder architectures such as the U-Net[11] for image segmentation tasks, it is an obvious approach to reuse some of those same principles except replacing convolutional operators with attention based operators. One such approach devised in this manner that has shown relative success is the UNETR network[5]. In practice, the UNETR network utilizes a ViT encoder to express features that are then used as input to a sequence of convolutional decoder blocks which are enhanced with dense skip-connections made up of intermediate ViT encoder layers.

We are interested in training networks with a similar configuration to those of UNETR. However, complications arise from the inclusion of skip-connections with a sequence parallel implementation due to the complex manner with which gradients are backpropogated. Therefore, in this work, we consider architectures similar to UNETR with the skip-connections removed. Using a network like this, we can determine effective approaches

[1]The specific process of distributing and tokenizing 3D image data into a sequential input vector will be detailed later in the ViT Implementation Details section.

to use partial self-attention outputs from a sequence parallel ViT encoder to segment images.

Sequence Parallel Strategies

Due to training being done in a sequence parallel manner, the output of a ViT encoder is a separate parallel self-attention output on each of the sequence parallel ranks. We are then interested in investigating effective approaches to provide input to the convolutional decoder. We want to investigate two scenarios: 1) the inputs to the convolutional decoder are partial self-attention outputs and 2) the partial self-attention outputs are gathered from across the sequence parallel ranks and concatenated to be used as input to the convolutional decoder. We name these scenarios the No-Gather and Gather approaches respectively. These two approaches have different strengths and weaknesses with respect to their computational performance, model accuracy, and ability to reproduce sequentially trained models.

In the No-Gather approach, each sequence parallel rank results in individual partial self-attention outputs, which are fed into separate convolutional decoders. This means that inputs to the convolutional decoder consist of only partial self-attention outputs that are derived from a portion of the input image. It is important to note that this is not equivalent to doing completely separate processing of data segments because we are still using the gradients that are averaged from across the sequence parallel ranks. Thus, the model weights are influenced by all of the different sources of partial self-attention and are kept in sync across the model. However, the result of the forward pass is a separate segmentation mapping on each sequence parallel rank for corresponding portions of input data. The drawback of this approach is that features from the whole data input are not used as input to the decoder. The strength of this approach is that it is computationally efficient because there are no additional necessary communications unlike the Gather approach (discussed next). Figure 2 depicts an example of this ViT encoder-convolutional decoder architecture using the No-Gather approach for a SeqPar=2 case. Although the architecture shown is compatible only for specific data sizes, i.e. where patches are of size $(16)^3$ and input data dimensions are a multiple of 16, the convolutional filters can easily be manipulated so that the output is transformed to be in the correct image space for the segmentation output.

The Gather approach, on the other hand, uses an all gather at the end of the ViT encoder to concatenate the partial self-attentions from across all sequence parallel ranks to achieve the full self-attention as shown in Equation 2. This means that when feeding the convolutional decoder, the full self-attention output is used so that segmentation output can be produced for the entire input image. The strength of this approach is that we are using the full self-attention output, so predictions are formed for the full input data. The weakness is that the all-gather introduces an extra communication to the forward pass of the neural network, so the computational performance will be worse than the No-Gather approach. A network like this is similar to the network configured in Figure 2 except there is an added concatenation of partial self-attentions so that the ViT encoder output is of size $l_x \times E_m$ and corresponding feature projection dimension is of size $\frac{W}{16} \times \frac{D}{16} \times \frac{H}{16} \times E_m$.

A factor of important consideration when using the Gather approach is that we need to correctly gather the partial self-attention output so that gradients are properly propagated in the backward pass during optimization. By design, communication operations do not inherently backpropagate gradients through an auto-differentiation framework. The reason for this is that the process is ambiguous and highly dependent on the specific operations and inputs. Therefore, we need to implement our own routines for handling the gradients. In our testing, we have found that it is necessary to perform specific operations during this all-gather to properly pass gradients to update model weights. In particular, we found that when all-gathering partial self-attention, scattering the gradients across the sequence parallel ranks is an effective approach to match the convergence results of non-sequence parallel training.

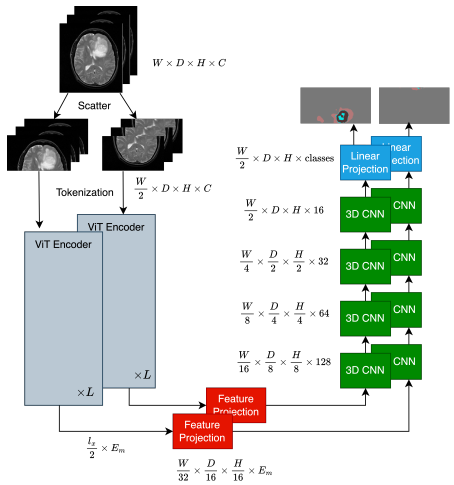

Figure 2. *A ViT Encoder-Convolutional Decoder network for the case of SeqPar* $= 2$ *where an Image is tokenized into patches such that* $l_x = \frac{W}{16} \times \frac{D}{16} \times \frac{H}{16}$*. Output of the ViT is projected back into image space and then fed into multiple 3D Convolution Layers (* $3 \times 3 \times 3$ *filters with Upsample Scaling of 2). The initial convolution layer maps from embedding length* E_m *to an arbitrary feature space chosen to be 128 for this example. This output is fed into a linear projection which transforms back into the original space based on the number of segmentation classes.*

Ultimately, the property that we are most interested in is the convergence of our sequence parallel training to networks trained without sequence parallelism. An approach that has this property provides validation that we can have similar training performance for all cases. This is particularly important for cases such as those with very long sequence input that can not be modeled except when using a sequence parallel approach.

Data

The current study uses the BraTS 2023 challenge dataset [8]. The BraTS 2023 dataset is a popular 3D brain tumor dataset that is often used as a benchmark for developing AI methods for tasks such as segmentation. The dataset consists of a collection of 1250 brain scans at $1mm^3$ resolution, or a total of $(240 \times 240 \times 155)$ voxels, for four different MRI modalities, including: T1-weighted images with and without contrast enhancement (T1 and T1-ce), T2-weighted (T2-w) images, and Fluid-attenuated Inversion Recovery (FLAIR) images. In the dataset, expert neuroradiologists meticulously reviewed and approved ground truth annotations for

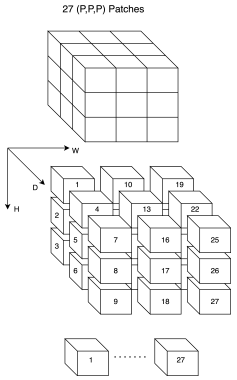

Figure 3. *A (W,D,H) image being tokenized into 27 (P,P,P) patches where $P = \frac{W}{3} = \frac{D}{3} = \frac{H}{3}$. Tokenized patches are ordered such that it first indexes through Height, then Depth, and then Width dimensions.*

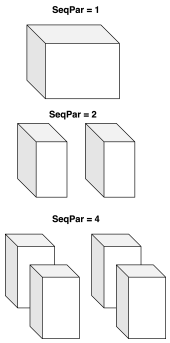

Figure 4. *The distribution of an image across different number of sequence parallel ranks using **spatial splitting**. After splitting, these sequence distributed images are each tokenized into patches similar to the process shown in Figure 3.*

each tumor subregion. The annotated tumor subregions are based on observed features visible to trained radiologists and include the Gd-enhancing tumor, peritumoral edematous/infiltrated tissue, and necrotic tumor core. The purpose of choosing this dataset is that it is a widely used 3D dataset that provide a large corpus of data for testing the performance of our methods. Particularly, we are interested in 3D data because it provides the most value with regard to training models with long sequence.

ViT Implementation Details

In order to more clearly show how 3D imaging data is processed by our sequence parallel ViT encoder, we present some specific implementation details. In this section, we first detail how 3D imaging data is arranged into patches and tokenized into sequence data that can be used as input data for ViT. Second, we will detail how 3D imaging data is split up between sequence parallel ranks.

Imaging data often contains too many pixels in order to be used as input directly to a ViT. For this reason, a patching strategy such as that introduced in [4] is typically used to provide a reduced form of representation of the data to be input into a ViT. We utilize a similarily devised strategy for our 3D data. For 3D images, we reshape the Image $\mathbf{x} \in \mathbb{R}^{W\times D\times H\times C}$ into a sequence of 3D patches $\mathbf{x}_p \in \mathbb{R}^{N\times(P^3\cdot C)}$ where (W,D,H) is the resolution of the original image, C is the number of channels, (P,P,P) is the resolution of each image patch, and $N = \frac{WDH}{P^3}$. Figure 3 depicts a scenario where an image is split into 3D patches such that $P = \frac{W}{3} = \frac{D}{3} = \frac{H}{3}$. An important implementation detail for these input patches lies within the reshaping of the image into flattened sequence of patches, specifically the ordering of those patches. For instance, in Figure 3 the sequence of patches are typically ordered by first cycling through the patches in the height dimension, then depth dimension, and then width dimension. While there exists schemes for incorporating a multiple dimension-aware position embedding to add more precise positional information to the embedding, for simplicity we use a one-dimensional position embedding.

When it comes to splitting up the image data to distributed data segments for the sequence parallel ViT encoder, there are a multitude of different possible configurations that we could derive for doing this for a given number of sequence parallel ranks. The simplest strategy we could use is to break them up by how they are ordered such as in Figure 3, which we call **ordered splitting**. However, we can also split them into regions such that patches are more spatially close together. Therefore, we also use an approach of splitting the patches evenly across ranks in the width, depth, and height dimension as the number of sequence parallel ranks increases. Figure 4 depicts three different setups we use for spatially distributing input images, which we call **spatial splitting**. These schemes are chosen in an attempt to keep input patches that are close in the image space closer together. It should be noted that the SeqPar $= 2$ case is the same in both ordered and spatial splitting. The orientation of the dimensional splitting can affect the accuracy of the model, particularly if there are certain important directional effects in the data. However, this should only affect the No-Gather approach, since the Gather approach retrieves the exact self-attention by concatenating partial attentions from across sequence parallel ranks.

Training Details

There are a myriad of different choices that can be made with regards to overall setup of the training that can affect the accuracy of the methods. In this work, we choose to keep most of these choices fixed to be able to compare between different model setups.

To prepare data for training, we use basic pre-processing routines to normalize data and crop background data from images. For the BraTS dataset, we consider background to be any zero valued voxel in the image. Particularly, these mostly come from edge of the MRI scans. This background data has very little labelling information and will only slow down model training with data that can easily be labelled with a minimum amount of inclusion.

Since the pre-processed training images are often too large to be used as input, most 3D image segmentation networks are not trained on a full image but rather on a subset of data which we call **tiles**. We use the terminology tiling in order to differentiate from the more commonly used terminology for this process - patching, which we use to describe the tokenization of input data to our ViT. In most of our experiments, we use tiles of size $(96)^3$ or $(128)^3$ as these are values that can be easily divided evenly into patch size of $P = 16,8,4$ that we are interested in studying to show the effects of patch size. These tile and patch sizes also

allow us to easily split up patches among sequence parallel ranks evenly. Ultimately, the goal is to provide a full segmentation mapping. Therefore, in order to provide a full segmentation mapping, a sliding window inference approach is used during validation to produce a full segmentation mapping from several tiles to fully assess the accuracy of a model.

The segmentation models we are interested in can become prohibitively expensive to train as the sequence length and model size grows. Therefore, an important aspect of our development process is to produce code that efficiently scales on multi-node, multi-GPU platforms. The platform we use to train our models is the Frontier supercomputer hosted by the Oak Ridge Leadership Computing Facility. Frontier has 9408 nodes, each consisting of 4 AMD MI250X, each with 2 Graphics Compute Dies (GCDs) and a total 64 GB of high-bandwidth memory per GCD, and a single 64-core AMD "Optimized 3rd Gen EPYC" CPU. Each of these GCDs can be defined logically as a single GPU, giving 8 logical GPUs per node. The main workhorse for our training are the AMD MI250X GPUs which we will use at minimum 8 logical GPUs per training run, i.e. to maximize the use of compute on a single node of Frontier.

The codebase[2] we use is extended from our previous sequence parallelism work in [13]. In this previous work, not only is an efficient sequence parallel approach for transformer models implemented, but it is also coupled with a scheme to incorporate data parallelism. Essentially, this scheme allows for two orthogonal levels of parallelism, sequence and data parallelism, to work together in a fashion to fully utilize the available hardware to scale up training. For example, if we were to train on a node of Frontier, which has 8 logical GPUs, with Sequence Parallelism of two, the added data orthogonal parallelism allows us to train 4 data batches simultaneously where each of the data parallel processes use two GPUs to perform the sequence parallel computations. For more details, we again refer the reader to [13]

In order to handle data loading and augmentation, we use the BatchGenerators library[3]. The BatchGenerators library provides a threaded CPU implementation for preparing data for training on GPUs. This implementation is particularly useful as it creates a pipeline of augmented data batches, which utilizes unused CPU resources, to be used by the GPUs since this process can often be a bottleneck for training. Specifically, the data augmentation we utilize includes rotations, elastic deformation, and random scaling. In this process, tiles are generated randomly from the cropped pre-processed training data. Therefore, there is no true notion of training on a full epoch of training data, i.e where every part of the training data is used in a deterministic fashion. Rather we choose to arbitrarily set an epoch to be defined as 50 iterations. This means that the number of tiles processed in an epoch in the results that follow can be calculated by the product of the number of iterations per epoch (50 in all our cases), the number of Data Parallel ranks, and the number of data batches per Data Parallel Rank.

The setting we choose for our optimizer and loss function are standard implementations often used for training segmentation networks. Particularly, we adopt many of the choices from model training done in [5]. For the loss function, we use a combination of soft dice[10] and cross-entropy loss. We train the model using an Adam Optimizer with initial learning rate of .0001 and a Cosine Annealing learning rate scheduler. For the experimental runs, we use floating precision throughout to avoid any complications that might arise when using lower precision operations provided through mixed precision training.

[2] `https://github.com/XiaoWang-Github/long-short-sequence-transformer/tree/seqpar_clean`

[3] `https://github.com/MIC-DKFZ/batchgenerators`

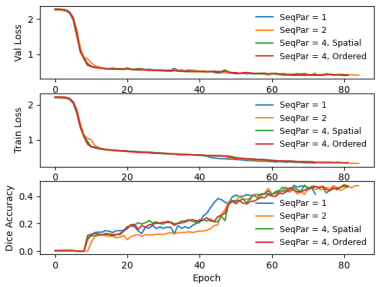

Figure 5. *Train loss, validation loss, and dice accuracy scores when training models using the Gather approach for handling the ViT encoder output.*

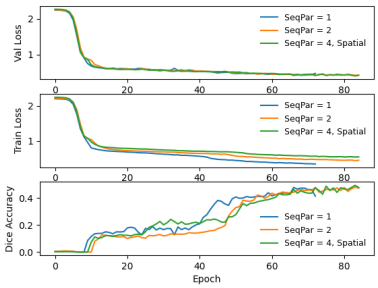

Figure 6. *Train loss, validation loss, and dice accuracy scores when using the No-Gather approach for handling ViT encoder output.*

Results

There are four main components to choose from when training these segmentation models: the size of ViT model (the number of layers L, the embedding Size E_m, and the number of Attention Heads A), the tile size T to be used, the patch size P with which to tokenize the data, and the number of sequence parallel ranks SeqPar. We use the following default settings: a ViT with $(L, E_m, A) = (12, 768, 12)$, $T = (96)^3$, and $P = (16)^3$. These settings will be used unless explicitly noted otherwise. The results to follow all use the same dataset with a 90:10 train/validation split where the model weights are all initialized with the same values. It should be noted that in all cases, the amount of data parallelism per run is the same, except when noted. This means that more GPUs are used when sequence parallelism increases in order to have the same amount of data throughput. Additionally, we maximize batch size per GPU in order to fully utilize the available GPU memory; this is set to 32 for the base case.

Experimental Results: Sequence Parallelism

The first result we look at is the performance of segmentation models with sequence parallelism compared to when the

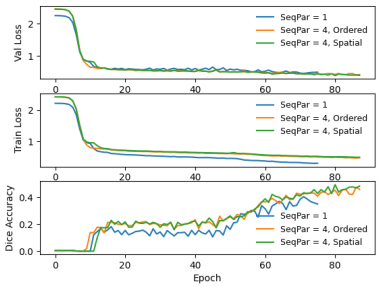

Figure 7. *Train loss, validation loss, and dice accuracy scores when using various splitting methods for the No-Gather approach for handling ViT encoder output.*

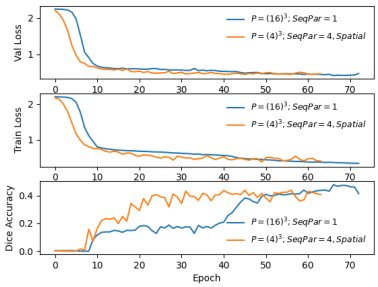

Figure 8. *Train loss, validation loss, and dice accuracy scores for training long-sequence ViT models.*

models are trained without sequence parallelism (SeqPar $= 1$). The importance of these results are to show that we see the correct training behavior as the number of sequence parallel ranks increases. Figure 5 shows results from training models using the Gather approach for handling ViT encoder output. As expected, the train and validation loss curves converge no matter the amount of sequence parallelism. Also, as expected, the order with which patches are sent to the sequence parallel ranks (Spatial vs Ordered splitting) does not change the performance of the networks.

Next, we look at the case where we use the No-Gather approach for the partial self-attention encoder features. This is an interesting case as it allows us to compare with the results from using the Gather approach to see whether the computational benefits of the fewer communications is worthwhile compared to the accuracy loss that is expected. Figure 6 shows results using the No-Gather approach for handling ViT encoder output. From the train loss curve, we can see that we clearly do not converge to the same values with the Gather approach (SeqPar $= 1$). However, the validation loss curves and dice accuracy scores follow similar patterns to those seen using the Gather approach. These results confirm that the No-Gather approach is viable, however more thorough studies need to be done to assess if this is true for all cases.

Lastly, we look at a case using the No-Gather approach comparing the two different splitting methods. In this case, we slightly modify the base problem to use $T = (128)^3$ and likewise reduce the batch size per GPU to 16 in order to accommodate the larger tile size. The results for this case can be seen in Figure 7. Again, in these results, we see that the training loss curves do not converge to the case without sequence parallelism. We also see that there is little difference between the two different splitting methods. It is not unreasonable to see the difference being small, however, we would not expect this behavior to continue as the amount of sequence parallelism is increased further.

Experimental Results: Long Sequence

To reinforce the main benefit of our work, which is the capability to train ViT based segmentation models with long sequences, we show some results where we train models that would not be able to be trained without sequence parallelism. Specifically, we train a model with $P = (4)^3$, which significantly increases the amount of tokenized patches used by the ViT. In this case, we needed to train with SeqPar=4 and a batch size per GPU of 1 in order to train a model without running out of GPU memory. Figure 8 shows these results compared to training with $P = 16$. This results show that, while decreasing the patch size does show faster convergence to higher dice accuracy scores, the maximum accuracy achieved after convergence are similar.

Unfortunately, through the process of training models with various combinations of model parameters, including increasing ViT model parameters, reducing patch size, and increasing tile size, we found that accuracy scores were always found to be in the range of $0.5 - 0.6$ in terms of dice accuracy. This issue most likely can be attributed to lack of training data available to create a generalizable model for segmentation.

Concluding Remarks

In this work, we introduced an efficient distributed sequence parallel approach for training transformer-based deep learning image segmentation models. We introduced two different approaches for using output form a sequence parallel ViT with a convolutional decoder. We show that through proper handling of the sequence parallel ViT encoder outputs with the Gather approach, we can train models with the same behavior as a model trained without sequence parallel model. We also showed that with the No-Gather approach we can provide similar quality of results, however the training of these models shows different convergence behaviors. Both of these approaches provide the capability of training of ViT based segmentation models using sequence lengths longer than typically possible.

Testing of these ViT segmentation models on longer sequence data did not provide higher accuracy for our given brain tumor segmentation dataset. Future efforts will involve the incorporation of several techniques to enhance performance and further investigate the utility of our sequence parallel approach. Techniques that will be considered include the involvement of pretrained models to incorporate learned features from other datasets that can possibly transfer their knowledge to the segmentation task. Other future work include the development of other features into our sequence parallel framework that have shown to be helpful for segmentation accuracy for other transformer-based implementations. These include the addition of patch merging layers to have multiple context levels within the model and the incorporation of skip-connections to the convolutional decoder to better inform these different context levels.

Acknowledgments

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan). The author Yuankai Huo would also like to acknowledge the support from the National Institute of Health under Grant R01DK135597.

References

[1] I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[2] R. Child, S. Gray, A. Radford, and I. Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[3] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[5] A. Hatamizadeh, Y. Tang, V. Nath, D. Yang, A. Myronenko, B. Landman, H. R. Roth, and D. Xu. Unetr: Transformers for 3d medical image segmentation. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 574–584, 2022.

[6] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

[7] N. Kitaev, Ł. Kaiser, and A. Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[8] H. B. Li, G. M. Conte, S. M. Anwar, F. Kofler, I. Ezhov, K. van Leemput, M. Piraud, M. Diaz, B. Cole, E. Calabrese, et al. The brain tumor segmentation (brats) challenge 2023: Brain mr image synthesis for tumor segmentation (brasyn). *ArXiv*, 2023.

[9] S. Li, F. Xue, C. Baranwal, Y. Li, and Y. You. Sequence parallelism: Long sequence training from system perspective. *arXiv preprint arXiv:2105.13120*, 2021.

[10] F. Milletari, N. Navab, and S.-A. Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 fourth international conference on 3D vision (3DV)*, pages 565–571. Ieee, 2016.

[11] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[12] A. Roy, M. Saffar, A. Vaswani, and D. Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

[13] X. Wang, I. Lyngaas, A. Tsaris, P. Chen, S. Dash, M. C. Shekar, T. Luo, H.-J. Yoon, M. Wahib, and J. Gounley. Ultra-long sequence distributed transformer. *arXiv preprint arXiv:2311.02382*, 2023.

[14] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Author Biography

Isaac Lyngaas received his BS in mathematics from South Dakota State University (2014) and his PhD in computational science from Florida State University (2018). Since then he has worked as a post-doctoral researcher and now a staff computational scientist at Oak Ridge National Laboratory. His work focuses on the development of AI methods with High Performance Computing.