Scaling Resolution of Gigapixel Whole Slide Images using Spatial Decomposition on Convolutional Neural Networks

ABSTRACT

Gigapixel images are prevalent in scientific domains ranging from remote sensing, and satellite imagery to microscopy, etc. However, training a deep learning model at the natural resolution of those images has been a challenge in terms of both, overcoming the resource limit (e.g. HBM memory constraints), as well as scaling up to a large number of GPUs. In this paper, we developed a distributed spatial decomposition method to train a Convolution Neural Network (CNN), on gigapixel images of size up to 49,152 x 49,152 pixels with 1,536 GPUs on the Summit Supercomputer. Our test dataset is from Whole Slide Imaging, where images can be in the size of 100,000 x 100,000 pixels or even larger. We demonstrated that this distributed decomposition method, even coupled with a simple CNN model, can yield competitive results compared to more complicated deep learning workflows such as multiple-instance learning. Moreover, our approach doesn't need pixel-level labels, since we're avoiding patching completely, and only slide-level labels are necessary. This is mainly achieved through a scalable implementation and performance optimization of the spatial decomposition method, and by leveraging the non-block fat-tree interconnect network of the Summit architecture, which enabled GPU-to-GPU direct communication.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning algorithms; Parallel algorithms; Distributed deep learning**.

KEYWORDS

Convolutional neural networks, spatial decomposition, medical imaging

ACM Reference Format:

. 2023. Scaling Resolution of Gigapixel Whole Slide Images using Spatial Decomposition on Convolutional Neural Networks. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (PASC '23)*. ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION AND BACKGROUND

The vast majority of computer vision applications in deep learning are based on low-resolution images such as 256 x 256 or smaller. For example, one of the most known computer vision datasets is the ImageNet [1] where its vast majority of the data is on a 256 x 256 scale, and the most common practice is to train it on a downscale size of 224 x 224. The size of the data samples will often drive the optimal choices for some of the architecture designs, for example, kernel size in convolution-based architectures, or how deep the model is. Also, the ability of a deep learning architecture to scale with the number of the model parameters is often measured with respect to the data size rather than the sample size, since a small number of applications acquire gigapixel-size images.

Motivation Scientific datasets differ from industry-based datasets, usually in terms of sparsity, but also a lot of times in terms of sample size, which they tend to be at a much larger pixel-size scale. Examples are from either observed data, such as microscopy or remote sensing imagery, certain types of medical imaging data, as well as simulated data, such as climate or cosmological simulations. The pixel size of those images can be 100,000 x 100,000 or even larger, and in some cases, the data comes in more than two dimensions. Additionally, as scientific instruments and supercomputers are advancing, data will be acquired in a finer and finer resolution, and therefore the deep learning workflows must develop algorithmic software-level parallelizations to process those images at their natural resolution.

Challenges Training deep learning models at the natural resolution of gigapixel-size images has been a challenge in terms of both, overcoming the resource limit (e.g. HBM memory constraints), as well as scaling up to a large number of GPUs. Also, most deep learning architectures and frameworks are optimized on lower-resolution images, therefore training a deep learning model to converge to a stable solution, on those extremely high-resolution images, and without losing pixel-level precision, is an active area of research.

Depending on the domain and the task at hand, these gigapixel-size images might have very large and very small features that need to be learned at the same time by the deep learning model. In this case, either the image needs to be downsampled to fit on a single device, and most likely the model won't be able to learn the small feature due to lowering the resolution. Or, if we don't downsample but instead divide the image in patches (or tiles), and so train the model on different patches, even though the small feature will be preserved the large feature might be split across devices and lose its content. That's why the model needs to be able to process the image as close as possible to its natural resolution.

We can consider three types of parallelism in deep learning, model parallelism, data parallelism, and sample parallelism. Of all three types, data parallelism has been explored the most while model parallelism seems to be on a high trend lately. Our focus here is on sample parallelism, which hasn't been explored as much as the other two, and the reason might be that it targets very specific use cases on very specific hardware, i.e. extremely high-resolution images, mostly scientific, on HPC systems. Also, the three are not mutually exclusive to each other, and theoretically applying them together will best optimize the overlaps between IO, compute, and

PASC '23, June 26–28, 2023, Davos, Switzerland

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

communication of a deep learning workflow. In practice, it is very challenging to orchestrate them efficiently to work together, and it is an active area of research

Work description In this work, we implement a distributed spatial decomposition method in Pytorch, by taking full advantage of the high GPU-to-GPU direct communications on a leadership High-Performance Computing (HPC) system. A detailed study of the performance was done on thousands of GPUs, and the convergence to a solution was tested on a known medical application with gigapixel-size images. Specifically, a dataset of Whole Slide Images (WSI) was used from The Cancer Genome Atlas (TCGA). WSI's are 2D digital microscopy images used by pathologists, and their resolution is often on the order of 100,000 x 100,000-pixel size. For the convergence to a solution tests, we trained on a 21,504 x 21,504-pixel size images, while we tested the performance of the method up to 49,152 x 49,152 pixels on 1,536 GPUs on the Summit Supercomputer, at the Oak Ridge National Laboratory.

A weakly-supervised classification task was performed on the WSI dataset, using the VGG16 architecture [2]. Even though a simple Convolution Neural Network (CNN) architecture was used, the implemented method is general enough so that convolutional layers from any deep learning architecture can be replaced by our spatial decomposed convolutions. The numerical output for the convolution layers should be the same between serial and distributed convolutional operations, while other layers, such as batch-normalization (not used in this study), will not. In that sense, the spatial decomposition method on CNNs acts more like one "super-GPU", using 1,536 GPUs in this case, for example, to process samples sequentially. In comparison, in the data-parallel approach, for example, the same model is copied across GPUs and so the data samples are processed in parallel.

Related work As far as we can tell, the idea of spatial decomposition in convolutions has been introduced by [3], where the concept is formally discussed and a small-scale study is performed. The [4] simulated the communication cost of all three types of parallelism and their combination. The [5] implement all three types of parallelism in the LBANN framework [6] and studied their weak and strong scaling behavior on individual layers, without looking at convergence to a solution of specific applications or full model architectures. [7] applied spatial decomposition using Mesh-TensorFlow on 3D computerized tomography images. Also, [8] applied sample parallelism on large satellite images for segmentation. More recent [9] developed a spatial decomposition approach using Pytorch-RPC, but also tested on convolution layers only, and without performing convergence on real applications.

The [10] work is the closest related to this study in terms of application and image size, where they use a RenNet-50 neural network on WSI images of size 21,500 x 21,500. Their approach to processing such large images leverages their custom unified memory library using TensorFlow, and they don't use any parallel methods to distribute the sample size. We tried to follow as closely as possible their setup and configuration parameters, in terms of both, WSI data scaling as well as the same split between training and validation.

Contributions We summarize our contributions as follows:

(1) Develop a new framework, using Pytorch, for spatial decomposition on convolutional neural networks, as described in details in Section 2. This capability enables:
   – Train CNN's on any image sizes at HPC.
   – Reaching five orders of magnitude samples-per-second improvement, using three orders of magnitude more compute resources, compared to a serial model with a CPU-GPU unified memory approach [10] while maintaining similar convergence to a solution. Detail differences between the two are described in Section 4.
   – Fewer annotations in computational pathology, since no pixel labels need it, and so exchange pathologist time for larger compute resources.
(2) Provide guidance, through the performance study shown in Section 3.2, of knobs and grid points to tune and consider when training and scaling gigapixel size images on HPC systems.
(3) Demonstrate that even a simple CNN, such as VGG16, can train and learn from gigapixel size images (shown on Section 4). As far as we can tell, this is the first time a simple model as VGG16 was shown to converge to a good solution on 21,504 x 21,504 pixel size images.

The rest of the paper is organized as follows. Section 2 describes in detail the method and the communication used. Section 3 describes the application used and presents performance results at scale. Finally, in Section 4 we present the results on the application used with our method.

2 SPATIAL DECOMPOSITION IN CNN

In this work we used a vanilla VGG16 Neural Network without batch normalization layers, so only Convolution (CONV), Fully Connected (FC), Pooling (POOL), and activation layers (RELU) were used. Something to consider when designing this framework is in which layers from our model we are going to apply spatial decomposition, understand the communication bottlenecks and take into account several factors, for example only the CONV and FC layers have learnable parameters while POOL and RELU layers don't, while other layers can be more complex to distribute across, like batch-normalization layers which we didn't include them in this initial work.

A typical deep learning workflow with CNN's starts by loading the model parameters to the GPU, then in the first forward pass, all intermediate outputs (we call those activations) from the model are saved. Then in the backward pass, we start from the end to calculate the gradients of the weights and biases while disregarding the forward activations as we go. Next, the optimizer will take a step to update the parameters based on the gradients, and then we will go to the next iteration to start computing the activations for the next inputs of data while the model weights and the gradients are still stored in memory.

2.1 Method Details

As we mentioned before the most common parallel strategy today in the deep learning community is data parallelism. And even though

we don't explore the hybrid of data parallelism and spatial decomposition in the work, it is worth describing the basic principles of it, since it is simpler and will help to better understand spatial parallelism. In the forward pass, batches are distributed among ranks, and the model weights are replicated in each rank. The calculation of activations is happening locally with no communication between ranks in the forward pass. In the backward pass, the calculation of gradients can also be done completely locally. But before the optimizer takes a step, a synchronous update needs to happen, using Allreduce and averaging all gradients across ranks. Pytorch has a specific function to wrap around the model to perform this step, and we use this same protocol to take the sum instead. Next, we describe our implementation of spatial decomposition.

A schematic view of the forward pass is shown in Figure 1. In our workflow, we first define a grid, where each grid point corresponds to a rank, and in our case, the number of spatial decomposition ranks matches the number of GPUs. Then each grid point has a description of each neighbor grid points, and the order we followed from its position on the grid is West-East-North-South. The next step is to set up the hallo dimensions; this is how many data points need to be transferred from the neighbor ranks according to the kernel dimensions used. Also, at this stage, we do the necessary zero padding in each grid point. Finally, we are doing the hallo exchange, first from West to East of every grid point and then from North to South, where neighbor grid data points are aggregated to make the tiles.

At this stage, a normal forward pass can start the compute in each tile, without any communication as it moves forward in the VGG layers, up to the final pooling layer. There we perform a max global pooling using an Allreduce communication across all ranks. The final layers are composition the classifier, where if they are deterministic they can be calculated independently in each rank. For example, that won't be the case if dropout layers are used, where it is a typical strategy to reduce over-fitting.

Then we can move to the backward pass, where a schematic view is shown in Figure 2. The process starts by calculating the gradients in the local data points plus the hallo regions already aggregated in the forward pass for the current layer parameters. A hallo exchange is also made in the backward pass of the gradients since the collective output gradients from the image are going to be used as input to the next layer. Finally, to finish the backpropagation pass and before the optimizer takes a step, a collective sum of all the partial gradients is performed from each rank. This last step is similar to how data parallel averages the gradients instead of summing them.

2.2 Communication

For the hallo exchange between all ranks, we used an all-to-all communication, and for the collective sum, we used Allreduce. Both were done through the Pytorch package, over NVIDIA/NCCL library, and for the most part of the deep learning workflow GPU-to-GPU communication was used. For the all-to-all we explicitly included it in the hallo exchange step, were for the Allreduce, we wrapped the model with Pytorch DDP, which defaults in a global average sum, and so we multiply the gradients by the number of

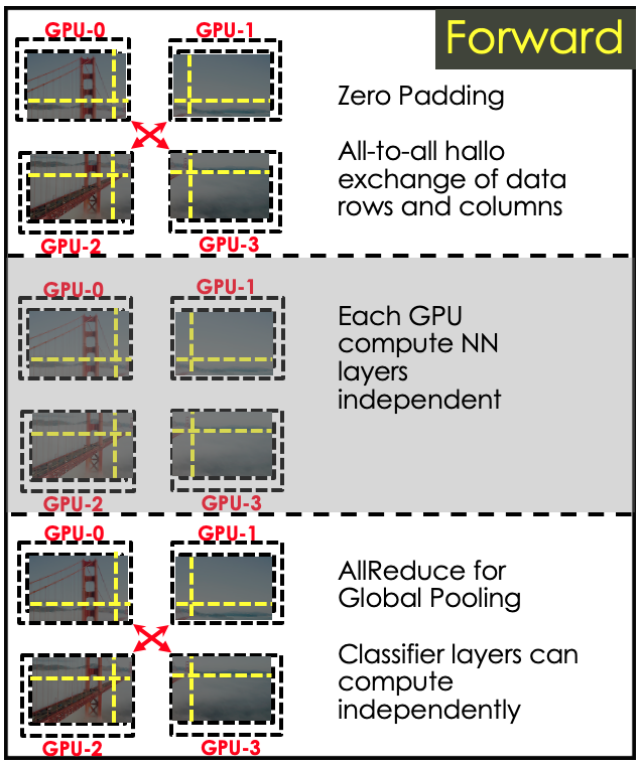

Figure 1: A schematic view of the spatial decomposition method in the forward pass of the training. The figure shows one image split into four equal tiles, on four GPUs, and a step-by-step description of the forward calculation from the implemented method perspective. Also, the zero padding and data rows and columns exchange are illustrated.

ranks to obtain the global sum. This was done using Pytorch's *register_hook*. Also, after all, spatial CONV layers have been computed and before the classification layers start, a global max pooling is performed using a local adaptive max pool layer of size 1 x 1, followed by a max Allreduce. For the classification layers, if there are serial layers and deterministic, we skipped Allreduce of gradients via internal parameter using Pytorch's *_ddp_params_and_buffers_to_ignore*. Also, the Pytorch-DDP flag *gradient_as_bucket_view* was used, to avoid the overhead of copying between gradients and Allreduce communication buckets.

3 APPLICATION USE-CASE

WSI images are digital microscopy images used by pathologists extensively in the last two decays and are acquired at a very high-resolution size. For example, a standard 4 x 6 cm glass slide at 40x magnification, after digitization turns into 200,000 x 300,000 image pixel in 2D. WSI images have been extensively studied using deep learning architectures, and they span a wide variety of tasks, from classification to segmentation to multi-tasking, and a lot of times are combined with pathologist reports and genomics data.

The most common strategy for applying deep learning on WSI images is by patching the image. This method works very well in

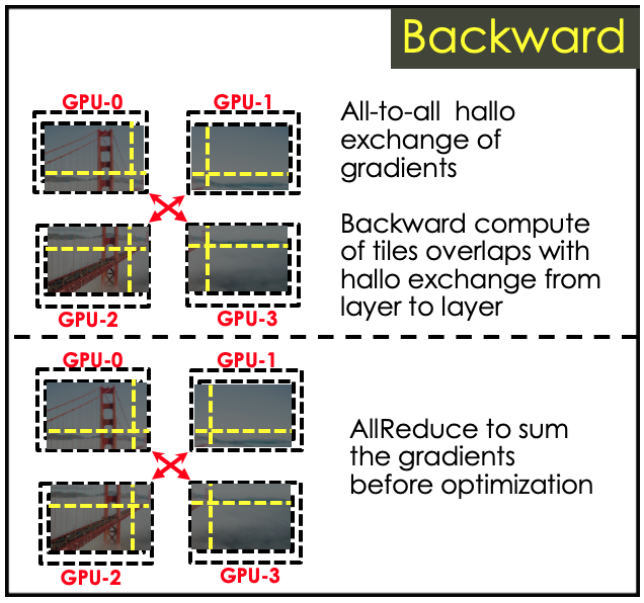

Figure 2: A schematic view of the spatial decomposition method in the backward pass of the training. This shows one image split into four equal tiles, on four GPUs, and a step-by-step description of the backward calculation from the implemented method perspective.

some cases, but there are some major limitations. The biggest one is that it requires pixel-level information, which can be very expensive and not realistic to have for the vast majority of WSI images. Also, it usually requires post-processing of the classification results, due to the large number of false positives it usually created.

Lately, the highest accuracy results on WSI images are achieved by applying multiple-instance learning (MIL) algorithms [11], in which a large size image is divided into multiple smaller images, called patches, of dimensions usually 256 x 256 pixels [12]. The final decision is made by a weakly-supervised training model on the extracted features from the image patches [13]. In most cases a 2D CNN is used to extract those patch features, usually either by a pre-trained model on ImageNet dataset [14] or patch images from The Cancer Genome Atlas (TCGA) dataset [15]. Some limitations of this method are that ImageNet pre-trained models may be limited due to the irrelevance of contents. Also the TCGA does not equip annotation of abnormality localization, thus the patch-level truth labeling is not intuitive. There are more resents self-supervised approaches [16], although the performance and the generality of those isn't as mature.

Also as we mentioned in the introduction this work [10] uses a ResNet-50 model on WSI images of size 21,500 x 21,500 pixels. They have a very detailed study of their model and among others, they perform transfer learning between different WSI domains and show performance results too. In their case, each sample was processed by a single GPU, and by using a unified memory approach they are leveraging the CPU memory to process such large images. With our method, we distribute each image sample in multiple GPUs, so we are not limited by the image size that we can run, and also we should expect much higher throughput numbers since once the data is loaded in the GPU, the rest of the training happening on the GPUs.

The current spatial decomposition approach workflow is more comparable to patching since we used a VGG16 model with our spatial distributed CONV and POOL layers. We also should be able to use more complicated workflows in the future, like the ones found in MIL or vision transformers, as long as they include CONV layers in their architecture. With that in mind, and comparing now with the patching method, in our case we don't need pixel-level information, since the whole image is used as input to the neural network, and so only slide-level information is needed it. Even though this approach is more computationally expensive in the training stage compared to the patching method, the fact that we don't need pixel-level information saves pathologists time from detailed label the data and can accompany them in the slide-level diagnosis. In the inference stage, where the model is mostly used, even though we haven't done a detailed study yet with our method, the time between spatial decomposition and the patching method should be at least equal since the slide for testing won't have pixel-level information, and all patches need to be inferenced from the model in both cases.

3.1 Dataset Details

In this work, we used one of the most commonly used datasets by the WSI community, i.e. The Cancer Genome Atlas (TCGA) non-small cell lung cancer (NSCLC) dataset. TCGA-NSCLC has a total of 1028 WSI slides and includes two sub-type projects, i.e., Lung Squamous Cell Carcinoma (TGCA-LUSC), with 512 slides and Lung Adenocarcinoma (TCGA-LUAD) with 516 slides. Following the same split between training and testing as well as scaling as in the [10], 831 slides were used for training and 197 for testing. Also following the same scaling for ×4 magnification to make 21,504 x 21,504-pixel images. To perform a weak scaling study we also generated samples of size 3,072 x 3,072, 6,144 x 6,144, 12,288 x 12,288, 24,576 x 24,576, and 49,152 x 49,152.

3.2 Performance Study

First, we convert the WSI slides into numpy arrays before the train starts. Then based on the desired grid, that was specified from the framework, the dimensions of the data points on each rank are provided, i.e. tiles. Then after having the dimensions of the tiles, and using the numpy's *mmap_mode*, each rank loads the part of the image that is going to process on the CPU, and augmentation is performed on those data points before the tile is loaded in the GPU. Also, we use 7 data loader workers in PyTorch for each rank, given that Summit has 6 GPUs per node and 42 physical cores.

Our training workflow also includes gradient accumulation so that we can use a larger global batch size, to further overcome some memory limitations, and converge to a stable solution. That means that our code is doing a few iterations of samples without updating the model variables. The gradients of those iterations are accumulated until we have configured to make a step, and only then model variables are updated. Also for the weak scaling studies below, we used a gradient accumulation of 4 and a local batch size of 4. Other optimizations include: FusedAdam optimizer, from

NVIDIA's APEX package, and mix precision so that we can enable the tensorcore use on NVIDIA's V100 cards.

The largest image that we can fit on the Summit V100 16GB cards, after our optimization, is 3,072 x 3,072 pixels. Figure 3 shows a strong scaling plot by measuring the throughput, starting from a 3,072 x 3,072-pixel image on a single GPU, and using the spatial decomposition method to distribute the sample among ranks. The ideal lines in the Figure are drawn from the single GPU and two node runs (i.e. 12 GPUs for Summit), where we multiply those numbers by the total ranks. And so ideal plots, by construction, are linear, as no change in communication or IO is assumed as we scale. At some point of course we will run out of data in some ranks. In the real case though communication and IO will add an overhead time on the throughput, and even though we should be able to hide IO behind compute, the many communication calls are expected to shape the performance.

As we can see in Figure 3 the performance is very good within nodes (i.e. 6 GPUs for Summit), but it drops as we scale to a larger number of nodes. This is expected since on Summit the GPUs within the node are connected with NVIDIA's NVLINK, which is faster than the node-to-node connection with dual-rail EDR infiniband and so the data needs to pass from the cross-socket bus on the node. However, as we scale up to pass the point of two nodes, there aren't any slower links in between and so it makes more sense to draw the ideal from two nodes. After the two nodes, the strong scaling is expected to increase, since each GPU has less data to compute, and we can see some marginal increases in Figure 3 as we move from two to sevel nodes. In practice, we lose some efficiency, and since deep learning is an optimization process, how does compute and communication interact and how efficient are their overlays and the load of each tiled worker matter a lot.

The same principles hold for the weak scaling plot as well, shown in Figure 4. We are drawing a constant image-per-second as ideal from both 1 node and 4 nodes, while emphasize the 4 nodes case since the single node GPUs have faster network links compared across nodes. For the same reason we start the plot from single node, instead of single GPU that we did in strong scaling. We run images of sizes 3,072 x 3,072, 6,144 x 6,144, 12,288 x 12,288, 24,576 x 24,576, and 49,152 x 49,152 on 1, 4, 16, 64, and 256 Summit nodes. We used a local batch size of 4 and a gradient accumulation of 4, since we want to cover as much data as possible in each iteration for a better overlay between compute and communication.

In both scaling plots, we've run with different grid decompositions of the full image. For example for a six-node run, i.e. 36 GPUs for the Summit supercomputer, we can make the grid as (6,6), (1, 36), or (36, 1), etc. Choosing the optimal grid dimension isn't straightforward, since this depends on the amount of data transfer and the ratio between fast and slow connections. Also, we have Allreduce communication that interacts between every rank in the row and column, and all-to-all in the hallo exchange portions of the workflow which will be mostly determined by the slowest link. For the latter, which is an easier case, if for example, we compare a 1x12 vs a 2x6 grid, the 1x12 has one halo exchange with a slow link, while the 2x6 has two halo exchanges, one with a slow link and one with a fast link. But we have to take into account the amount of data exchange and Allreduce cost, which is more complex to calculate as well as the overlays with the compute.

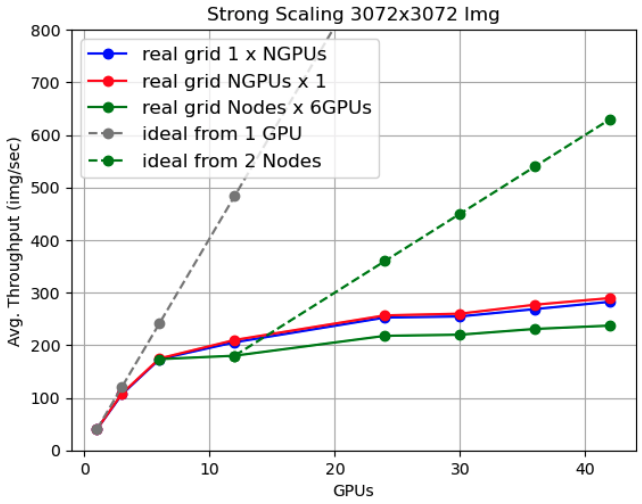

Figure 3: A strong scaling plot of the spatial decomposition method. The size of the images was scaled to 3,072 x 3,072 for each sample. The full training dataset was used for each data point in the plot. Starting from a single GPU and then gradually splitting the image amongst ranks, as more GPUs are used. The ideal plots are drawn from single GPU and two nodes (12 GPUs) runs. For the blue curve, the grid is made as *1 x number of GPUs*, for the red curves, the grid dimension is made as *number of GPUs x 1*, while for the green curve the grid as made as *number of Nodes x 6*.

Some general trends from the plots can be made, such as the *2 x number of GPUs/2* is slower than the *Nodes x 6GPUs* and *number of GPUs/2 x 2*. But as we can see in the strong scaling as we have fewer data per rank the trend can change between those three grid modes. An auto-tuning mechanism, before each run starts, might be very helpful for optimizing the correct configuration given the many parameters involved. Also for the weak scaling, we used *2 x number of GPUs/2* instead of *1 x number of GPUs* configuration used in the strong scaling since we run out of data in some tiled ranks for the 256 node runs (1,536 GPUs).

4 RESULTS

As we mentioned before, our model is a VGG16, without batch normalization layers. The last average pooling layer was replaced by a global max pooling, whereas as it was observed from here [10], the model converges to higher classification accuracy. A dropout layer of 0.5 was also used before the classification layers. We used image sizes of 21,504 x 21,504 pixels with a square grid of 42 x 42 and a total number of 1,764 GPUs for 2.5 hours. The local batch size was set to 1 and a gradient accumulation step of 18. The initial learning rate was set to 2e-05, with a weight decay of 2e-06, as well as a reduction on plateau scheduler with patience of 5 epochs and a reduction factor of 0.1. For data augmentation, we followed [10] as it was described, and so we did flipping, random contrast (multiplication by 0.5–1.5), brightness (multiplication by 0.65–1.35), and hue (addition by 32–32), but in our case, we didn't perform any rotation or translation. Also, the augmentation was done on the fly,

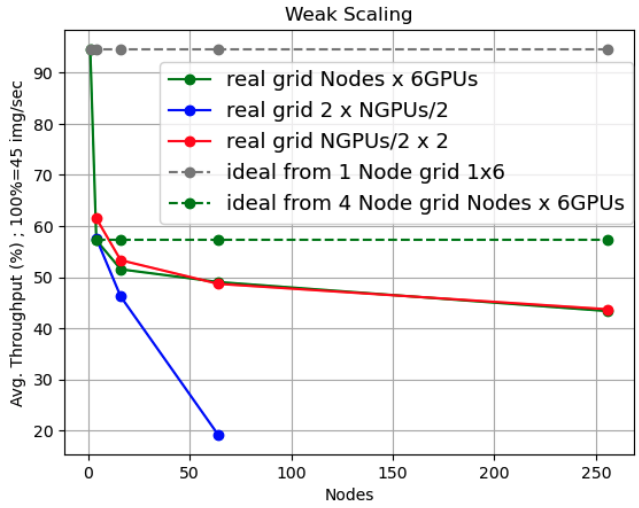

Figure 4: A strong scaling plot of the spatial decomposition method. The full training dataset was used for each data point in the plot. We start from one node and scale the image samples to 3,072 x 3,072 pixels, and then for four nodes, we scale each sample to 6,144 x 6,144 pixels. This scaling factor of four in the resources is used because of 2D pixel images. We scale further to 12,288 x12,288, 24,576 x 24,576, and 49,152 x 49,152-pixel sizes, on 16, 64, and 256 Summit nodes. The ideal plots are drawn from single-node and four-node runs. For the blue curve, the grid is made as *2 x number of GPUs/2*, for the red curves, the grid dimension is made as *number of GPUs/2 x 2*, while for the green curve the grid as made as *number of Nodes x 6.*

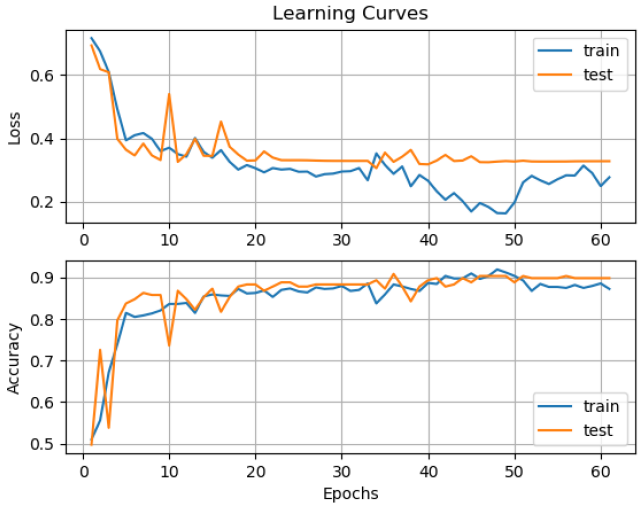

Figure 5: Learning curves of the spatial decomposition method, applied on TCGA-NSCLC dataset, processing one sample per iteration of pixel size 21,504 x 21,504 using 1,764 NVIDIA V100 GPUs for 3.5 hours. The blue curves are the loss and the classification accuracy on the training dataset, while the orange curves are the loss and the classification accuracy on the testing dataset. The *x-axis* is the number of iterations from the full training and test dataset.

where each rank applied augmentation on its specific tiles. Finally, the weights of VGG16 were initialized from ImageNet, and as it has been observed by the community, it boosts the performance by at least a 10% accuracy increase, compares to training with random weights initialization.

Figure 5 shows the learning curves of loss and accuracy for the training and testing dataset. We get a 90% classification score accuracy on the test dataset after 60 epochs. In comparison [10] reaches a 93% accuracy after 200 epochs on the same training and test datasets, using ResNet50 (a deeper CNN model than VGG16). In this work we haven't performed an extensive search of hyperparameters, nor optimized the neural network architecture. Our focus of this study were the correctness of the method and stable convergence runs of a known application, so the classification accuracy score exhibited in Figure 5 could be sub-optimal. Nevertheless, the 90% accuracy is highly competitive against other studies on the same dataset, reported around 80~90% of accuracy scores [17, 18].

As was mentioned in the contributions, we achieved five orders of magnitude improvement, using the presented spatial decomposed framework on 1,764 GPUs, compared to a serial model on a single GPU, using GPU-CPU unified memory approach. Specifically, we used a VGG16 on 21,504 x 21,504-pixel size images, on 1,764 NVIDIA V100 16GB cards, with Pytorch-1.10, CUDA-11.0, and NCCL-2.11.4 reaching 86 images-per-second. In comparison, [10] reports a 0.002 images-per-second using ResNet-50 on 21,500 x 21,500-pixel size images, on 1 NVIDIA V100 32GB card with the Tensorflow framework. Even though the deep learning models are different, literature shows that image-per-second between VGG and ResNet are very similar [19], with the latter achiving higher accuracy consistently on the ImageNet dataset.

5 CONCLUSION

In this work, a general method was presented for learning directly from gigapixel-size images using a distributed spatial decomposed CNN. The presented method can scale to arbitrary pixel size images and reach good throughput numbers, as we scale up, by taking full advantage of the very dense communication network that leadership HPC facilities offer. On the other hand, showing converges to a good solution, as resolution scales up, is an active area of research, and here we showed that even a simple VGG16 model can get competitive results. Even though we used a specific model, any type of neural network that includes convolutional layers can use our framework to process gigapixel-size matrices in those layers. Additionally, this method can be expanded on larger than 2D data, which most likely can benefit even more from this type of parallelism due to much larger memory footprints and the icreasing complexity of performing patching on larger dimensions.

While the application use-case was on medical imagining, this method can be applied to any high-resolution images of this size. As HPC resources become richer and more widely available than before, and as scientific instruments keep increasing the data acquisition resolution we believe that these types of methods would be more widely used in the future. Finally, deep learning model

parallel approaches are dominating the natural language processing space at the moment, while in computer vision the need of such large models is still in search. Maximizing the image resolution size processed by deep learning models might help with model scalability in computer vision in the future, having deeper deep learning model to take advantage of the finer resolution.

ACKNOWLEDGEMENTS

This research was sponsored by and used resources of the Oak Ridge Leadership Computing Facility (OLCF), which is a DOE Office of Science User Facility at the Oak Ridge National Laboratory supported by the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

REFERENCES

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[3] Peter Jin, Boris Ginsburg, and Kurt Keutzer. Spatially parallel convolutions, 2018.

[4] Amir Gholami, Ariful Azad, Peter Jin, Kurt Keutzer, and Aydin Buluc. Integrated model, batch and domain parallelism in training neural networks. 2017.

[5] Nikoli Dryden, Naoya Maruyama, Tom Benson, Tim Moon, Marc Snir, and Brian Van Essen. Improving strong-scaling of cnn training by exploiting finer-grained parallelism, 2019.

[6] Brian Van Essen, Hyojin Kim, Roger Pearce, Kofi Boakye, and Barry Chen. Lbann: Livermore big artificial neural network hpc toolkit. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, MLHPC '15, New York, NY, USA, 2015. Association for Computing Machinery.

[7] Le Hou, Youlong Cheng, Noam Shazeer, Niki Parmar, Yeqing Li, Panagiotis Korfiatis, Travis M. Drucker, Daniel J. Blezek, and Xiaodan Song. High resolution medical image analysis with spatial partitioning, 2019.

[8] Sudip K. Seal, Seung-Hwan Lim, Dali Wang, Jacob Hinkle, Dalton Lunga, and Aristeidis Tsaris. Toward large-scale image segmentation on summit. In *49th International Conference on Parallel Processing - ICPP*, ICPP '20, New York, NY, USA, 2020. Association for Computing Machinery.

[9] Aristeidis Tsaris, Jacob Hinkle, Dalton Lunga, and Philipe Ambrozio Dias. Distributed training for high resolution images: A domain and spatial decomposition approach. In *2021 IEEE/ACM Redefining Scalability for Diversely Heterogeneous Architectures Workshop (RSDHA)*, pages 27–33, 2021.

[10] Chi Long Chen, Chi Long Chen, Chi-Chung Chen, Wei-Hsiang Yu, Szu Hua Chen, Yu Chan Chang, Tai I. Hsu, Michael Hsiao, Chao-Yuan Yeh, and Cheng Yu Chen. An annotation-free whole-slide training approach to pathological classification of lung cancer types using deep learning. *Nature Communications*, 12(1):1193–1193, 2021.

[11] Oded Maron and Tomás Lozano-Pérez. A framework for multiple-instance learning. *Advances in neural information processing systems*, 10, 1997.

[12] Ming Y Lu, Drew FK Williamson, Tiffany Y Chen, Richard J Chen, Matteo Barbieri, and Faisal Mahmood. Data-efficient and weakly supervised computational pathology on whole-slide images. *Nature Biomedical Engineering*, 5(6):555–570, 2021.

[13] Kausik Das, Sailesh Conjeti, Abhijit Guha Roy, Jyotirmoy Chatterjee, and Debdoot Sheet. Multiple instance learning of deep convolutional neural networks for breast histopathology whole slide classification. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 578–581. IEEE, 2018.

[14] Jiawen Yao, Xinliang Zhu, and Junzhou Huang. Deep multi-instance learning for survival prediction from whole slide images. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 496–504. Springer, 2019.

[15] Abtin Riasatian. Kimianet: Training a deep network for histopathology using high-cellularity. Master's thesis, University of Waterloo, 2020.

[16] Richard J Chen, Chengkuan Chen, Yicong Li, Tiffany Y Chen, Andrew D Trister, Rahul G Krishnan, and Faisal Mahmood. Scaling vision transformers to gigapixel images via hierarchical self-supervised learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16144–16155, 2022.

[17] Zhuchen Shao, Hao Bian, Yang Chen, Yifeng Wang, Jian Zhang, Xiangyang Ji, et al. Transmil: Transformer based correlated multiple instance learning for whole slide image classification. *Advances in Neural Information Processing Systems*, 34:2136–2147, 2021.

[18] Jingwei Zhang, Xin Zhang, Ke Ma, Rajarsi Gupta, Joel Saltz, Maria Vakalopoulou, and Dimitris Samaras. Gigapixel whole-slide images classification using locally supervised learning. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 192–201. Springer, 2022.

[19] Muhammed Mutlu Yapıcı, Adem Tekerek, and Nurettin Topaloğlu. Performance comparison of convolutional neural network models on gpu. In *2019 IEEE 13th International Conference on Application of Information and Communication Technologies (AICT)*, pages 1–4, 2019.