Evaluation of Scalability for Distributed Data-Parallel Training of Swin Transformer V2

by

Dillon Garrett

Bachelor of Science in Computer Engineering, University of Pittsburgh, 2021

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science

University of Pittsburgh

2023

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

Dillon Garrett

It was defended on

April 27^{th}, 2023

and approved by

Ahmed Dallal, PhD, Assistant Professor, Department of Electrical and Computer Engineering

Samuel Dickerson, PhD, Associate Professor, Director, Department of Electrical and Computer Engineering

Thesis Advisor: Alan D. George, PhD, Mickle Chair Professor, Electrical and Computer Engineering

Evaluation of Scalability for Distributed Data-Parallel Training of Swin Transformer V2

Dillon Garrett, M.S.

University of Pittsburgh, 2023

As recent research demonstrates, the trend in model size across deep learning has rapidly increased, helping to further the state-of-the-art. Along with an increase in model size comes increased computational demands on hardware and software computing platforms, leading to training scalability being of interest. Following the development of transformer-based models, it has become common practice to begin training with a pre-trained model and fine-tune it on a specific dataset to allow for wider adoption without full model retraining. While originally designed for the natural language processing field, transformers have been adapted to many other domains. Swin Transformer V2 is a transformer model used for computer-vision tasks that achieved state-of-the-art semantic segmentation results. This research provides a scalability analysis for the distributed data-parallel training of Swin Transformer V2 on the semantic segmentation vision task. The ADE20K semantic segmentation dataset is used for training instances to fine-tune this model. A weak scalability experiment is designed, increasing the number of GPUs for training while holding the problem size constant. To implement this experiment, the sub-batch size per GPU is held constant at 8 images per GPU per iteration and the total number of iterations is scaled down. Training time, GPU utilization, and CPU utilization metrics for single- and multi-GPUs are measured on NVIDIA A100 SXM, NVIDIA A100 PCIe, and NVIDIA V100 PCIe GPU platforms hosted by the Center for Research Computing at the University of Pittsburgh. Training speedup and parallel efficiency metrics are calculated. For all computing platforms, training on 2 GPUs is 26% faster on average when compared to single GPU training. However, diminishing returns are observed when adding additional GPUs because smaller speedup benefits are observed. When increasing the number of GPUs from 2 to 4, the training is only 1.9% faster on aver-

age on NVIDIA A100 PCIe and NVIDIA V100 PCIe nodes. For NVLINK-enabled NVIDIA A100 nodes, training is only 2.9% faster when increasing the number of GPUs from 4 to 8. Consequentially, distributed data-parallel training of Swin Transformer V2 scales poorly as the number of devices is increased.

Table of Contents

List of Tables

List of Figures

Preface

This work was supported by the NSF SHREC industry and agency members and by the IUCRC Program of the National Science Foundation under Grant No. CNS-1738783.

1.0 Introduction

In recent years, the transformer-based computer vision model, Vision Transformer (ViT) [10], has achieved notability for common vision tasks as an alternative to traditional convolutional neural network (CNN) models. By introducing a novel image pre-processing technique, ViT enables the use of transformers with multi-headed self-attention modules for vision tasks. The original ViT and its successors have achieved state-of-the-art (SOTA) accuracy on tasks such as image classification [10] and semantic segmentation [23, 22].

With the difficulty of deep-learning tasks ever growing and the significant increase in computing power available came the trend of significantly larger model sizes. A significant jump in model size leads to the increased complexity of training scalability, due to increased demands on the requirements of training steps such as batch processing and backpropagation. Small-scale distributed GPU clusters are often the only option available to academic groups for the acceleration of deep learning training, making them of interest for this work.

In this thesis, a scalability analysis is performed for the distributed data-parallel training of Swin Transformer V2 [22]. The computer vision task chosen to train on is semantic segmentation, a common vision application for fine-tuning. A current Swin Transformer V1 [23] for semantic segmentation implementation is adapted to support Swin Transformer V2. For continuity, the ADE20K semantic segmentation dataset [39] employed in the Swin Transformer V1 paper is used. Models are programmed using PyTorch and are trained on single- and multi-GPU computing platforms. Training scalability is analyzed via training time, speedup, parallel efficiency, GPU utilization, and CPU utilization.

In Chapter 2, related work in the field of deep learning is discussed. Chapter 3 reviews more specific background information relating to this research. The next chapter covers the approach including computing platforms and implementation details. Chapter 5 provides an evaluation of the results. Chapters 6 and 7 consider conclusions and potential for future work expanding upon this research.

2.0 Related Work

This related work chapter is split into three sections. First is an overview of convolutional neural-network models for vision tasks. Next is an overview of popular transformer-based models. Finally, recent transformer models for image classification and semantic segmentation are reviewed.

2.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) have consistently dominated computer vision tasks. Deng et al. released ImageNet, a large-scale vision dataset, in 2009 pushing researchers to develop considerably larger and more powerful models due to the challenge of learning 1,000 classes for the image classification task [8]. Krizhevsky et al. used the ImageNet dataset to train AlexNet, the first deep CNN model to achieve SOTA image classification [20]. Training AlexNet's 60M parameters required employing model parallelism 2 NVIDIA GTX 580 GPUs. Each model-parallel training instance took 5 to 6 days of training time. Other popular CNN architectures inspired by the work of AlexNet include GoogLeNet [33], VGG [31], ResNet [18], and EfficientNet [34].

CNN-based models are a common choice for semantic segmentation. Long et al. introduced FCN, which adapts the classification models AlexNet, VGG-16, and GoogLeNet into segmentation models to achieve SOTA performance on segmentation [24]. Major changes include discarding the final classification layer, replacing the rest of the fully connected layers with convolutional layers, and fine-tuning pre-trained weights for semantic segmentation. Each model is then combined with the FCN architecture for improved performance on the finer-grained segmentation task. Other CNNs for segmentation include U-Net [29], Mask R-CNN [17], Fast R-CNN [15], and YOLO [28].

2.2 Transformers

In recent years, the race to develop deep-learning models to achieve maximal accuracy and performance has led to the widespread use of transformers, with the popularization of models such as GPT-3 [12]. Vaswani et al. introduced the transformer architecture for natural language processing (NLP), offering an alternative to CNNs and other machine translation models by utilizing the self-attention mechanism [36]. To train the original 213M parameter transformer, 300K training steps were performed over 3.5 days on 8 NVIDIA P100 GPUs. Transformers saw SOTA results on language modeling tasks [36, 27, 9]. Radford et al. offered a new training method for the original transformer combining generative pre-training with discriminative fine-tuning [27]. Pre-training the model on a significantly larger text dataset before fine-tuning it for a specific task led to a new SOTA in language understanding tasks. Devlin et al. released BERT, integrating masked language modeling into model pre-training to continue improvements on transformer-based architectures [9].

Along with the introduction of transformers came a significant increase in the parameter count of the SOTA models [37]. This trend in model size leads to significantly larger computational demands on complex, distributed hardware computing systems and increased complexity of software algorithms for model pre-training and fine-tuning. Larger models require longer training instances and put more demand on device memory.

2.3 Vision Transformers

While dominant in NLP, transformers have also become ubiquitous in vision applications. Transformer-based vision models have recently demonstrated SOTA capabilities, offering a compelling alternative to the standard CNN vision model. Dosovitskiy et al. introduced a novel data pre-processing technique with ViT allowing for the use of a pure transformer for image recognition [10]. By breaking an image up into 16 x 16 patches and flattening each one, the transformer encoder can process the input in a similar manner to any NLP application. While this breaks the paradigm of arbitrary input sizes, this encoding scheme

allows the vision transformer to process image data. This encoding scheme additionally alleviates the issue of every pixel attending to every other pixel across the image, with ViT performing self-attention within patches. ViT has model sizes ranging from 86M to 632M parameters. Improvements upon ViT came with other classification models including DeiT [35], BeiT [1], DINO [38], and MAE [16].

3.0 Background

This chapter is separated into three sections. First, distributed model training paradigms and distributed computing metrics are discussed. Second, an overview of the Swin Transformer V1 and V2 models is given. Lastly, the semantic segmentation computer vision task and ADE20K dataset are reviewed.

3.1 Distributed Model Training

In this section, distributed GPU computing, parallel computing metrics, and distributed model training are discussed. Distributed GPU computing is used for the acceleration of the training for deep learning models. GPUs use a single instruction, multiple data architecture, enabling their use for parallel programming by performing many calculations at once. They are commonly programmed via the host-device method, with a CPU as the host computer and a GPU as the device. For an extra level of parallelization, GPUs can be combined via distributed computing methodologies, where work can be distributed across each GPU and performed concurrently. Model training is a well-suited application for distributed GPU computing due to the common use of massively parallel calculations and the upward trend in model size.

The NVIDIA Tesla V100 PCIe [7], NVIDIA Ampere A100 40 GB PCIe [6], and NVIDIA Ampere A100 40 and 80 GB SXM [6] GPU architectures are of interest for this research. Architecture, memory, and interconnect specifications for these GPUs are in Table 1. It is of note that the cores on the V100 use the NVIDIA Volta architecture, which is the predecessor to the Ampere architecture. NVIDIA Tensor cores are designed for a specific fused multiply-add operation commonly used in the training of neural networks. Tensor cores on Ampere devices can provide up to 20X higher performance than Volta Tensor cores [6]. While V100 PCIe architecture has the most Tensor cores, the Volta Tensor cores are significantly outperformed by the Tensor cores on each Ampere GPU. The GPU interconnects of interest are

Table 1: GPU architecture specifications of [7, 6] are listed.

GPU	**CUDA Cores**	**Tensor Cores**	**Memory (VRAM)**
V100 PCIe	5,120	640	32 GB
A100 PCIe	6,912	432	40 GB
A100 SXM 40 GB	6,912	432	40 GB
A100 SXM 80 GB	6,912	432	80 GB
GPU	**Memory BW**	**Memory Clock**	**Mem. Bus Size**
V100 PCIe	900 GB/s	1.75 Gbit/s	4096-bit
A100 PCIe	1,555 GB/s	2.4 Gbit/s	5120-bit
A100 SXM 40 GB	1,555 GB/s	2.4 Gbit/s	5120-bit
A100 SXM 80 GB	2,039 GB/s	3.2 Gbit/s	5120-bit
GPU	**Interconnect**	**Interconnect BW**	**GPU Microarch.**
V100 PCIe	PCIe 3.0 $\times$16	32 GB/s	Volta
A100 PCIe	PCIe 4.0 $\times$16	64 GB/s	Ampere
A100 SXM 40 GB	NVLINK 3.0	600 GB/s	Ampere
A100 SXM 80 GB	NVLINK 3.0	600 GB/s	Ampere

PCIe [19] and NVIDIA's NVLINK [25]. NVLINK is a GPU-to-GPU interconnect that offers multiple paths between each GPU, all encompassed in a mesh network. PCIe can be used for any device-to-device interconnect, but uses a more rigid bus structure by moving data over lanes. The NVLINK interconnect enables data transfers between GPUs at significantly faster rates than PCIe.

Data parallelism and model parallelism are two common methods for accelerating model training. In data-parallel training, models are replicated on each device (GPU), data is split-up and distributed to each device, and resulting weights are aggregated together during the backward pass. In the PyTorch DistributedDataParallel package [21], weights are aggregated through an all-reduce operation using NVIDIA NCCL, GLOO, or MPI as the backend to communicate over the GPU interconnect. In contrast, model parallelism divides the layers of a model over all devices and creates a pipeline of data connecting each device to perform training. This pipeline is used to pass data from device to device during the forward pass and send weights from device to device during the backward pass. In this work, the focus is on the use of data parallelism for model training. For more details on data- and model-parallel training in PyTorch, see [21].

When parallelizing an application, a change in runtime often occurs due to work being spread across multiple processors as well as the overhead of communication. The ratio of single-processor runtime to multi-processor (parallel) runtime is known as speedup. In this research, the speedup of interest is single- versus multi-GPU training time. This relationship is denoted by Equation 3.1, where T_s is the time spent training a model on a single GPU, T_p is the time spent training the same model on multiple GPUs, and p is the number of GPUs used. In this case, speedup gives insight into the scale at which training time changes as the number of GPUs is increased. Parallel efficiency, shown in Equation 3.2, is the ratio of speedup to the number of GPUs. Parallel efficiency is a crucial metric to analyze this application's scalability, as it denotes the extent to which the extra devices are being utilized to improve training performance.

$$Speedup = \frac{T_s}{T_p} \tag{3.1}$$

$$Parallel\ Efficiency = \frac{Speedup}{p} \tag{3.2}$$

When considering speedup and parallel efficiency, it is important to acknowledge ideal values for each metric. For a training instance on N GPUs, the ideal speedup is achieved when the speedup is equal to the number of GPUs, N. Ideal parallel efficiency is reached in that same case, with an ideal value of 1. The closer the speedup is to N and the closer the parallel efficiency is to 1, the better the parallel implementation is.

In this research, device utilization is considered to be the ratio of the amount of compute time spent for a specific device, T_d, to the total compute time, T_t. This relationship is seen in Equation 3.3. This metric is displayed as a percentage for better understanding. Device utilization offers an internal view of the amount in which an algorithm uses each device. In this work, device utilization is used to evaluate the extent to which the GPU(s) and CPU are employed in each training iteration.

$$Device\ Time\ Utilization = \frac{T_d}{T_t} \tag{3.3}$$

3.2 Swin Transformer V1 & V2

With the introduction of ViT for image classification, transformer-based vision models have been adapted for semantic segmentation. Liu et al. present Swin Transformer V1 and V2, achieving SOTA performance for both object detection and semantic segmentation challenges [23, 22]. This model makes two key advancements on the original ViT implementation, the first being shifted-window attention. While patchwise global attention is much faster than performing global attention across an image, it is a compute bottleneck for ViT. Swin Transformer offers an alternative by stacking-window and shifted-window self-attention mechanisms, reducing computational complexity from quadratic ($O(n^2)$) to linear ($O(n)$) with respect to the number of image patches. To improve performance on semantic

segmentation, this model introduced patch merging, a technique to create hierarchical feature maps through merging and pooling image patches between Swin Transformer Block layers. This convolution-like technique allows for Swin Transformer to perform well on fine-grained prediction tasks, such as segmentation. For Swin Transformer V2 [22], changes include using post-layer normalization and using scaled-cosine attention. For training on semantic segmentation, Liu et al. used MMEngine, MMCV, and MMSegmentation, a training engine and vision libraries developed by OpenMMLab [3, 2, 4]. These libraries utilize the distributed data-parallel packages in PyTorch for multi-GPU training [11].

3.3 Semantic Segmentation

Semantic segmentation is the process of assigning a class label to each pixel within an image. When compared to image classification or object detection, segmentation is a fine-grained computer vision problem. This task is of interest due to its high compute and memory requirements and application in fields such as autonomous vehicles, remote sensing, and medical imaging and diagnoses [14, 30, 32]. Zhou et al. released the ADE20K semantic segmentation dataset in [39], which consists of 25,000 images with a variety of scenes and objects from 150 semantic categories. Images are also labeled for the classification task. The 25,000 images are split up into 20,000 in the training set, 2,000 in the validation set, and 3,000 in the test set. Each image is annotated pixel-by-pixel with class labels. In total (images and labels combined), the ADE20K dataset requires about 2.3 GB of memory for storage. The accuracy metric used for this dataset is the mean intersection-over-union (mIoU), or the Jaccard index, which is commonly used in semantic segmentation benchmarks [14, 5]. The mIoU is calculated as the mean of the intersection between predicted and actual segmentation values for each pixel over the union of said values. Further details on this metric are described in [39].

4.0 Approach

This approach chapter contains two sections. In the first section, an overview of the hardware and software computing platforms is provided. In the next section, the distributed data-parallel training of Swin Transformer V2 implementation is discussed.

4.1 Computing Platforms

With the goal of analyzing the scalability of distributed data-parallel training, there was an interest in platforms with next-generation hardware and multi-GPU compatibility. Compute nodes are hosted by the Center for Research Computing (CRC) at the University of Pittsburgh. For the Swin Transformer V2 training implementation, CRC V100, A100, and A100_NVLINK partitions were chosen. Exact specifications for each partition can be found in Table 2.

The CRC V100 testbed consists of 1 node with 4 NVIDIA V100 32 GB GPUs, dual-socket Intel Xeon Gold 6126 CPUs, and multi-GPU capability via a PCIe Gen 4.0 interconnect. Each V100 GPU has 5,120 CUDA cores, 640 Tensor cores, and 32 GB of HBM2 VRAM [7]. On the CRC A100 testbed, there are 12 nodes each with 4 NVIDIA A100 40 GB GPUs connected by PCIe Gen 4.0 with multi-GPU capability, 2 of which have dual-socket Intel Xeon Gold 5220R CPUs and the rest with single socket 2nd-Gen AMD EPYC 7742 CPUs. The CRC A100_NVLINK testbed has 2 nodes each with 8 NVIDIA A100 80 GB SXM GPUs and 3 nodes with 8 NVIDIA A100 40 GB SXM GPUs, all with multi-GPU capability and coupled through an NVLINK switch. Each node has dual-socket 2nd-Gen AMD EPYC 7742 CPUs. There are 6912 CUDA cores, 432 Tensor cores, and either 40 GB or 80 GB of HBM2 VRAM on each A100 GPU [6].

Table 2: Platform specifications of compute nodes at [13] are listed.

CRC Platform	**Nodes**	**GPU**	**# GPU**
V100	1	NVIDIA V100	4
A100	12	NVIDIA A100	4
A100_NVLINK	5	NVIDIA A100 SXM	8
CRC Platform	**GPU Mem.**	**Mem. BW**	**GPU Interconnect**
V100	32 GB	900 GB/s	PCIe 3.0
A100	40 GB	1,935 GB/s	PCIe 4.0
A100_NVLINK	40 & 80 GB	1,555 & 2,039 GB/s	NVLINK 3.0
CRC Platform	**Interconnect BW**	**CPU**	**Node RAM**
V100	32 GB/s	Intel Xeon Gold 6126	192 GB
A100	64 GB/s	AMD EPYC 7742	512 GB
A100_NVLINK	600 GB/s	AMD EPYC 7742	1024 GB

To ensure software compatibility, Singularity v3.9.6 is utilized alongside a Docker container for the software environment on CRC. The PyTorch development Docker image used has PyTorch v1.12.0, Python v3.7.13, CUDA v11.3, and cuDNN v8.2.0.53 [26]. For OpenMMLab libraries, MMCV version 1.4.7 and MMSegmentation version 0.11.0 are used, compiled via GCC 7.5 for CUDA 11.3.

4.2 Distributed Data-Parallel Training of Swin Transformer V2

For the implementation of distributed data-parallel training of Swin Transformer V2, the Swin-Transformer-Semantic-Segmentation repository from [23] was used in this research. The training was performed on the ADE20K [39] dataset, as used in the original Swin Transformer work [23]. At the time of writing, this repository only supported semantic segmentation on Swin Transformer V1. To continue development on this work, support for

semantic segmentation on Swin Transformer V2 was developed and added to this repository. The model specifications in PyTorch for Swin Transformer V2 released with [22] were adapted for use in this semantic segmentation repository. Swin Transformer V2 achieved SOTA semantic segmentation on ADE20K, increasing upon the previous Swin Transformer's SOTA by +6.3 mIoU [22], making it of interest for model training analysis. In this research, the SwinV2-base model with 88M parameters was implemented.

Training is performed via the distributed data-parallel (DDP) paradigm. For this implementation, the PyTorch DistributedDataParallel package was used [21] in conjunction with MMEngine, MMCV, and MMSegmentation [3, 2, 4]. In an instance of DDP training on N devices, models are duplicated onto each device. For each iteration, a batch of data is loaded on the CPU and split N-ways. Each sub-batch is then sent to its respective GPU. Training then continues as normal with forward and backward passes on each device in parallel. To complete each iteration, gradient losses are synchronized via an all-reduce operation and the optimizer on each device calculates the final gradient sum for each weight. NVIDIA NCCL's implementation of the all-reduce operation is used in this application. For a more in-depth explanation of the PyTorch DDP package, see [21]. With this algorithm, there is potential for speedup resulting from the distribution of work across multiple devices. However, it is important to note there is significant communication overhead due to data sharing and gradient synchronization.

To investigate the scalability of model training, a weak scaling experiment is designed. With weak scaling, problem size is held constant as the number of devices is increased. To mimic the results from [23], the base problem size of 160K training iterations with 8 images per iteration sent to a single GPU is used. When the number of devices is increased, the number of iterations is varied while the number of images sent to each GPU during each iteration is held at 8. This ensures that the SwinV2-base model sees the exact same amount of data during each training instance. Normally there are validation steps and checkpointing steps throughout a training instance to track and save progress. Checkpointing and validation steps are removed to ensure the work being performed is solely the work intended for this experiment. More detailed specifications for this experiment are shown in Table 3.

Table 3: Weak scalability experiment specifications.

Number of GPUs	**Number of Iterations**	**Images per GPU per Iteration**
1	160K	8
2	80K	8
4	40K	8
8	20K	8

The crucial metric for analyzing scalability for this experiment is the training time for each training instance. This metric denotes the potential for scaling the number of GPUs for improved training performance. Training time is gathered for each experiment specification on the CRC V100, CRC A100, and CRC A100_NVLINK compute nodes. Due to the limit of 4 GPUs per node on CRC V100 and CRC A100, training time for 8 GPUs on these platforms is not gathered. Timing metrics represent the wall clock time starting from before the first training iteration to the end of the last iteration.

From the training time, speedup and parallel efficiency are calculated. Speedup is calculated as the ratio of the training time when using a single GPU on a specific node to the training time when using N GPUs on that same node. For parallel efficiency, the ratio of the aforementioned speedup to the number of GPUs is calculated.

After collecting training time data, data for the GPU and CPU utilization metrics are gathered. For this data, the PyTorch Profiler module is utilized [11]. PyTorch Profiler offers functionality to track CPU and CUDA (GPU) time. As the profiler takes a significant amount of time to run, only 8 iterations are run for each experiment configuration, still holding the images per GPU per iteration at 8. To include a buffer for device warm-up time, GPU and CPU time are measured over 8 iterations. From the CPU and GPU time over 8 iterations, CPU and GPU utilization metrics are calculated.

5.0 Evaluation

The evaluation chapter is comprised of four sections. First, training time metrics are displayed and notable results are discussed. In the second section, speedup and parallel efficiency metrics are analyzed. In the next section, device utilization metrics are shown. Finally, platform comparison figures are presented and hardware comparisons are drawn. Computing platform specifications are shown in Table 2 and experiment specifications details can be found in Table 3. Due to GPUs per node constraints, training time metrics for CRC A100 and CRC V100 are gathered only up to 4 GPUs.

5.1 Training Time

Training time benchmarks are displayed in Figure 1. This metric is the primary method for the evaluation of training scalability. Units for training time are in hours with no training job shorter than one day. Training is fastest on the CRC A100_NVLINK platform and slowest on the CRC V100 platform. There is a notable difference in single GPU training time on CRC A100 versus CRC A100_NVLINK, even with these platforms utilizing the same GPU architecture. This difference is likely due to the communication benefits of the NVLINK versus PCIe, as NVLINK offers significantly faster data transfer rates over its interconnect. The CRC A100_NVLINK platform also boasts memory advancements that make memory access significantly faster than that on CRC A100. Another key finding determined from this metric is the leveling effect noticed as the number of GPUs is scaled from 2 to 4 for PCIe-based platforms and from 4 to 8 for NVLINK-based platforms. For CRC A100 and CRC V100, there is an average of 1.3% reduction in training time when scaling from 2 to 4 GPUs. As for CRC A100_NVLINK (80 GB and 40 GB), a reduction of 2.9% on average as the number of GPUs is increased from 4 to 8 is seen. This leveling-off effect in training time

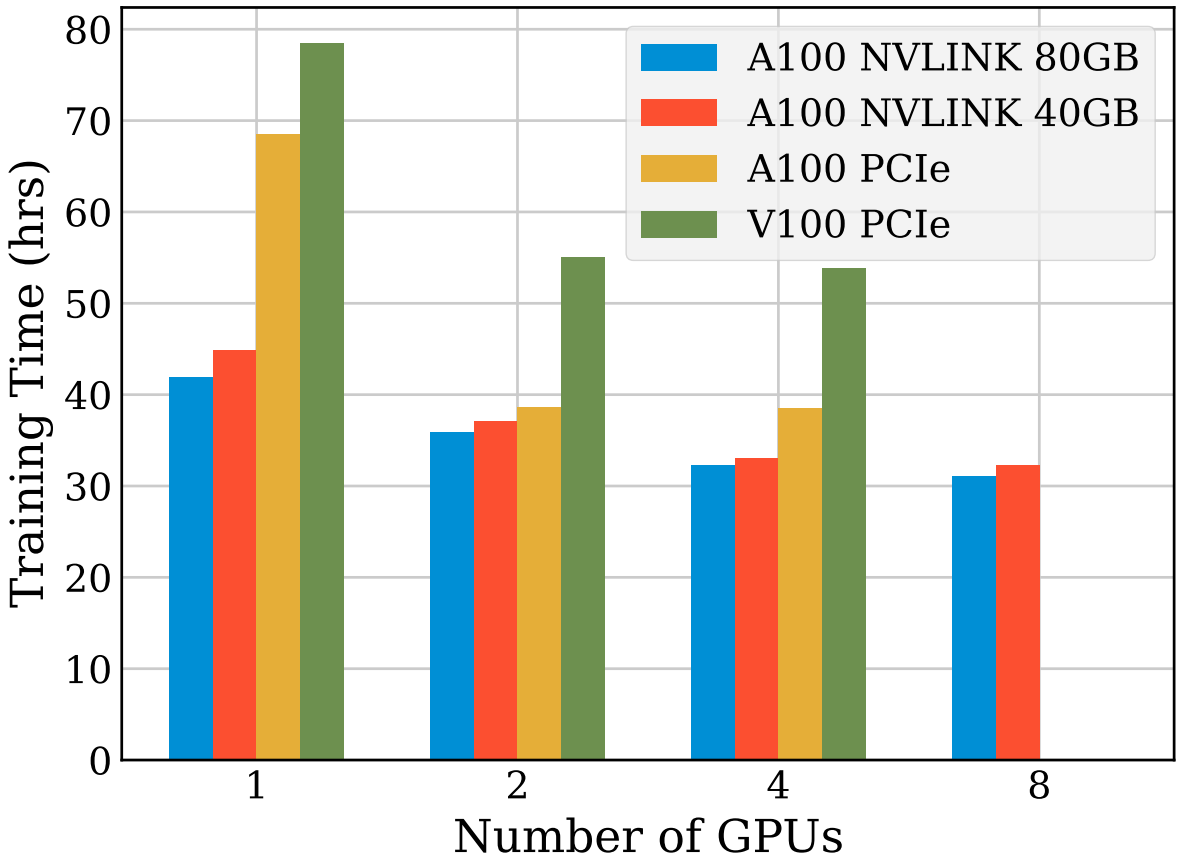

Figure 1: Training time versus number of GPUs used for Swin Transformer V2 distributed data-parallel training. Notable 34% reduction in single GPU training time from CRC A100 to CRC A100_NVLINK (40 GB) due to performance gain from NVLINK. Experiment specifications for training runs are in Table 3. Error is negligible at this time scale.

reduction is significant when compared to the 26% reduction in training time as the number of GPUs is scaled from 1 to 2 GPUs. With this result, the claim that training scalability on this specific dataset with this specific model significantly drops off at higher device counts can be made.

5.2 Speedup and Parallel Efficiency

Speedup metrics are displayed in Figure 2. A dashed line chart is used to emphasize the trend in speedup as the number of GPUs is scaled up. Each line has three points to show speedup at 1, 2, and 4 GPUs. Due to the limit of 4 GPUs per node on CRC A100 and CRC V100, the x-axis on this plot stops at 4 GPUs. Note the high speedup values for CRC A100 and CRC V100 for 2 and 4 GPUs. These higher speedup values for the PCIe-based nodes are primarily due to the significantly worse performance on single GPU training for these nodes and can be deemed outliers. All platforms realize a significant speedup in training time (1.4× on average) from single- to 2-GPU training, with the lowest being CRC A100_NVLINK at a value of 1.17×. However, speedup values for each hardware platform begin to plateau when training on 4 GPUs, showing diminishing returns as the number of GPUs is increased. This trend is especially noticed on PCIe-based platforms. Speedup on CRC V100 goes from 1.43× to 1.46×, and speedup on CRC A100 goes from 1.75× to 1.8×, a mere 1% difference on average. Comparing speedup values at 4 GPUs to ideal speedup (4×) shows that the DDP training algorithm for Swin Transformer V2 on ADE20K is significantly underperforming when scaling up.

Parallel efficiency is another key metric for scalability analysis, displayed in Figure 3. Again, a dashed line chart is used to demonstrate the trend in parallel efficiency as the number of GPUs is scaled up. Each line is shown with three points to demonstrate parallel efficiency at 1, 2, and 4 GPUs. The x-axis on this plot is halted at 4 GPUs due to the limit of

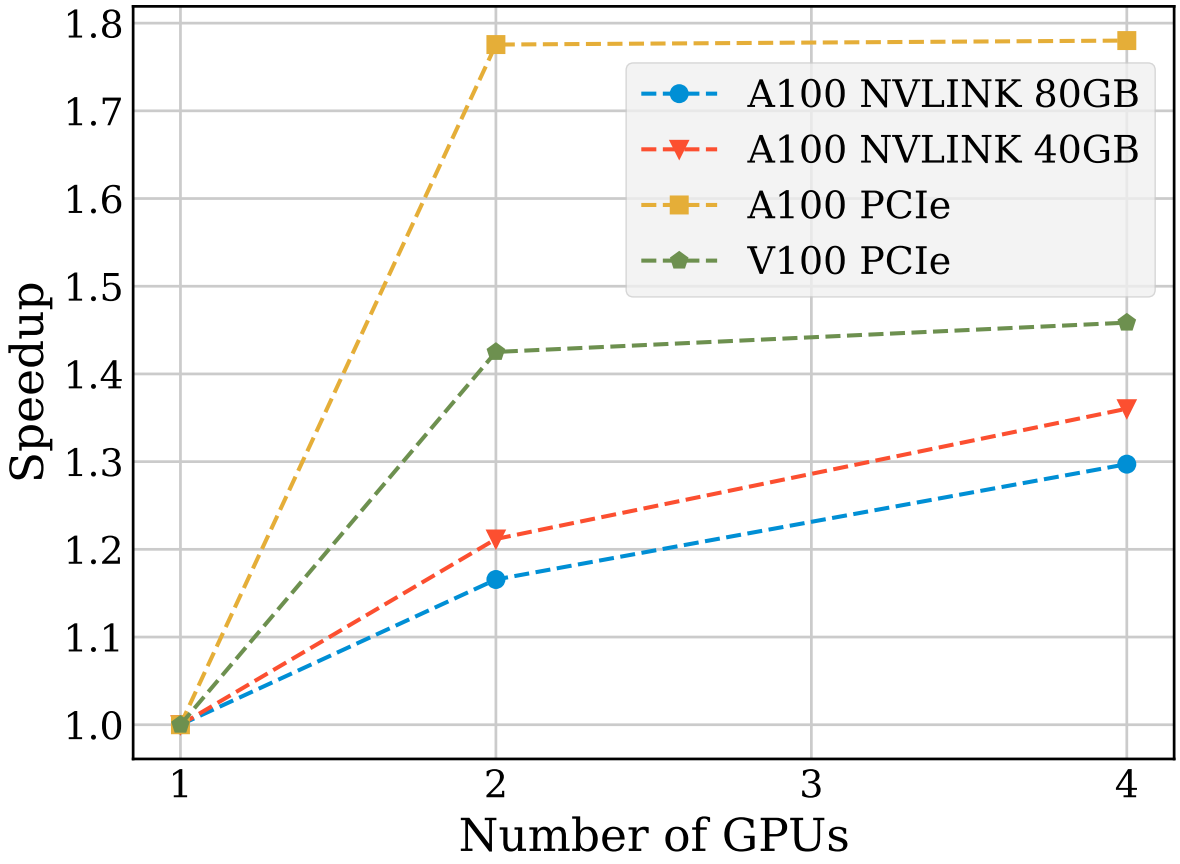

Figure 2: Speedup versus number of GPUs used for Swin Transformer V2 distributed data-parallel training. Due to significantly slower single GPU training times, CRC A100 and CRC V100 demonstrate inflated speedup values. See Table 3 for experiment specifications.

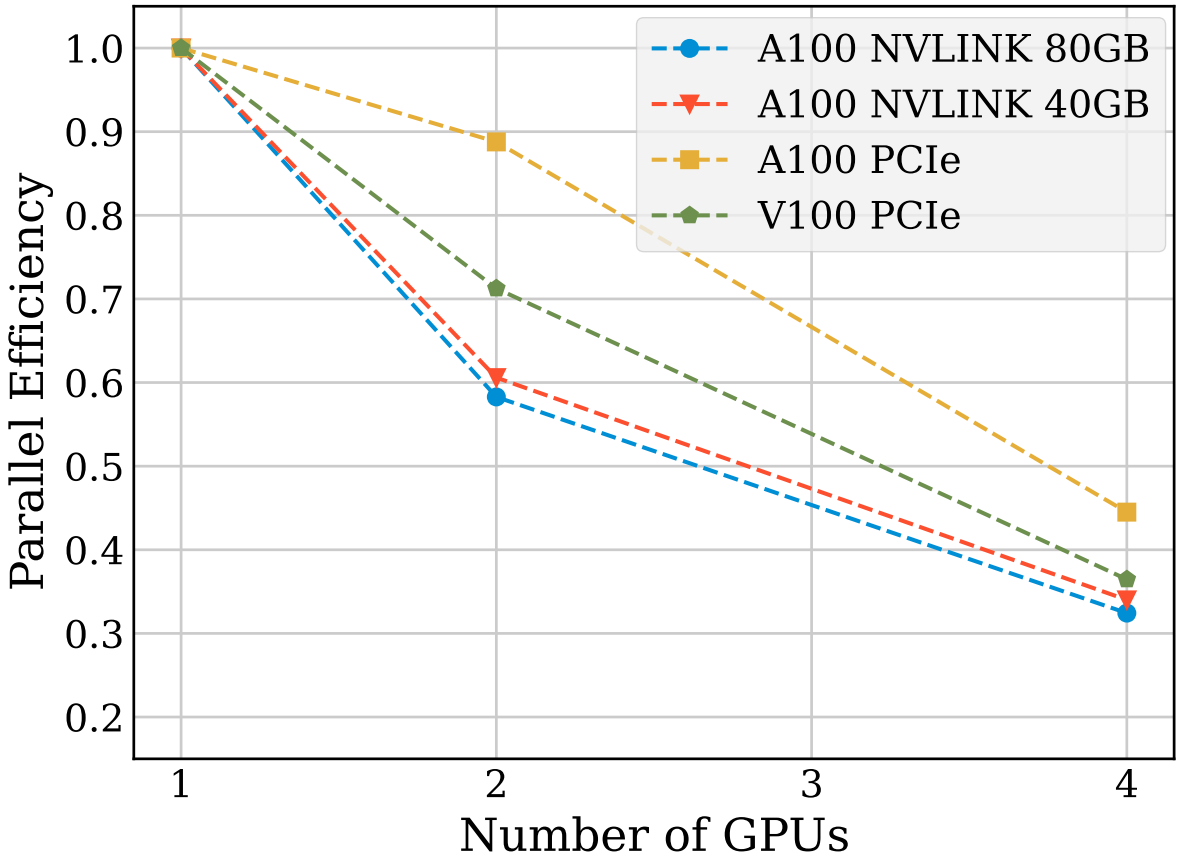

Figure 3: Parallel efficiency versus number of GPUs used for Swin Transformer V2 distributed data-parallel training. See Table 3 for experiment specifications.

4 GPUs per node on CRC A100 and CRC V100. It is of note that at 4 GPUs, all platforms achieve sub-0.5 parallel efficiency. This falls well below ideal parallel efficiency. A major takeaway from this outcome is that extra devices are significantly underutilized during DDP training of this specific model on this dataset at higher scaling levels (4 GPUs).

To observe closer-to-ideal speedup and parallel efficiency metrics, parallel algorithms require significant optimization. The DDP training algorithm used in this research does not utilize the parallel programming concept of communication and computation overlap. Both the host and device are relegated to wait until the other is finished working to continue with training. This concept can be introduced to DDP training by overlapping work from the CPU and GPU. For example, while the CPU is waiting for the GPU to complete forward and backward passes during each iteration, the next batch of data could be prepared on the CPU. The CPU could then send out to the GPU via asynchronous communication methods, enabling the GPU to receive new data right when the previous iteration is complete.

5.3 Device Utilization

The final metrics studied are GPU and CPU utilization. GPU utilization and CPU utilization plots are shown in Figure 4 and Figure 5, respectively. Both GPU and CPU utilization are percentage-based metrics. The y-axis on the GPU utilization plot ranges from 0% to 40% to further distinguish each point. Note the difference in GPU and CPU utilization for each specific compute platform as the number of GPUs is increased. GPU utilization has a downward trend for each platform, while CPU utilization has an upward trend for each platform. While the GPU time slightly increases as the number of devices is scaled up, a significant increase in CPU time is observed, causing these opposite trends for GPU and CPU utilization. The increase in CPU time can be attributed to the extra work demanded from data loaders on the CPU, as 8 images are sent out per iteration to each GPU.

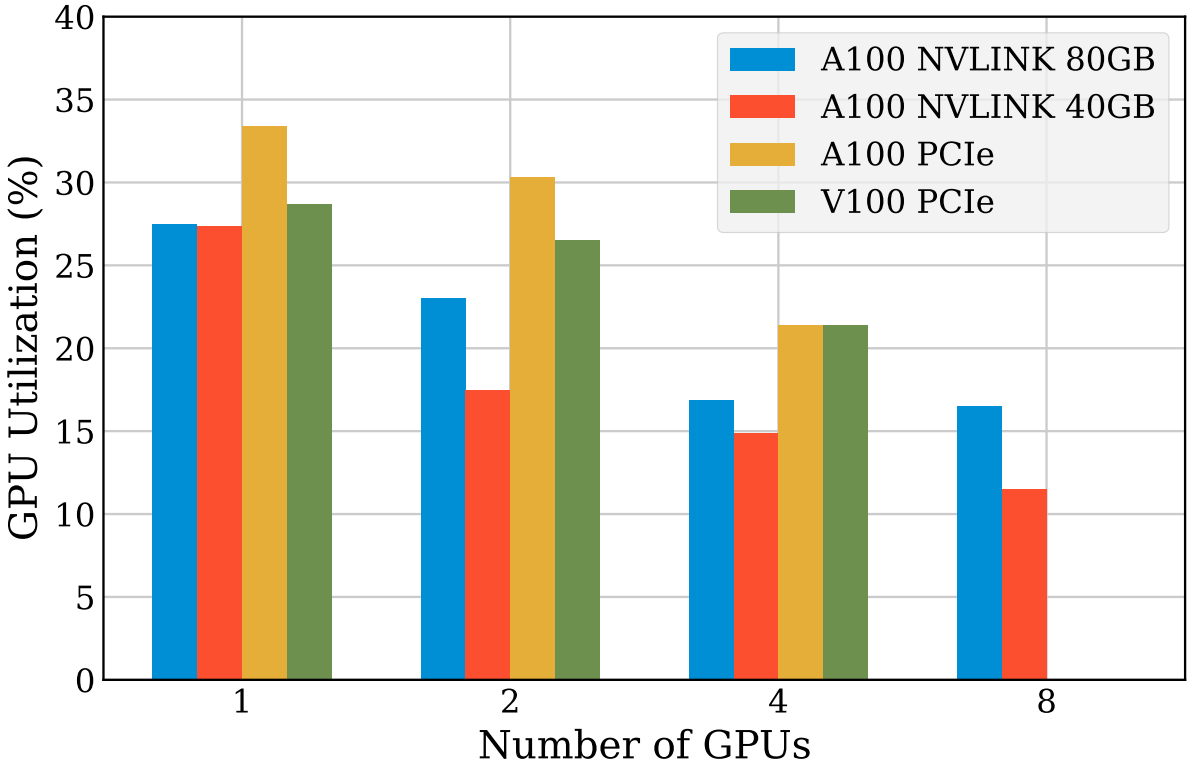

Figure 4: GPU Utilization versus number of GPUs used for Swin Transformer V2 distributed data-parallel training. Metrics gathered via PyTorch Profiler. Error is negligible compared to averages.

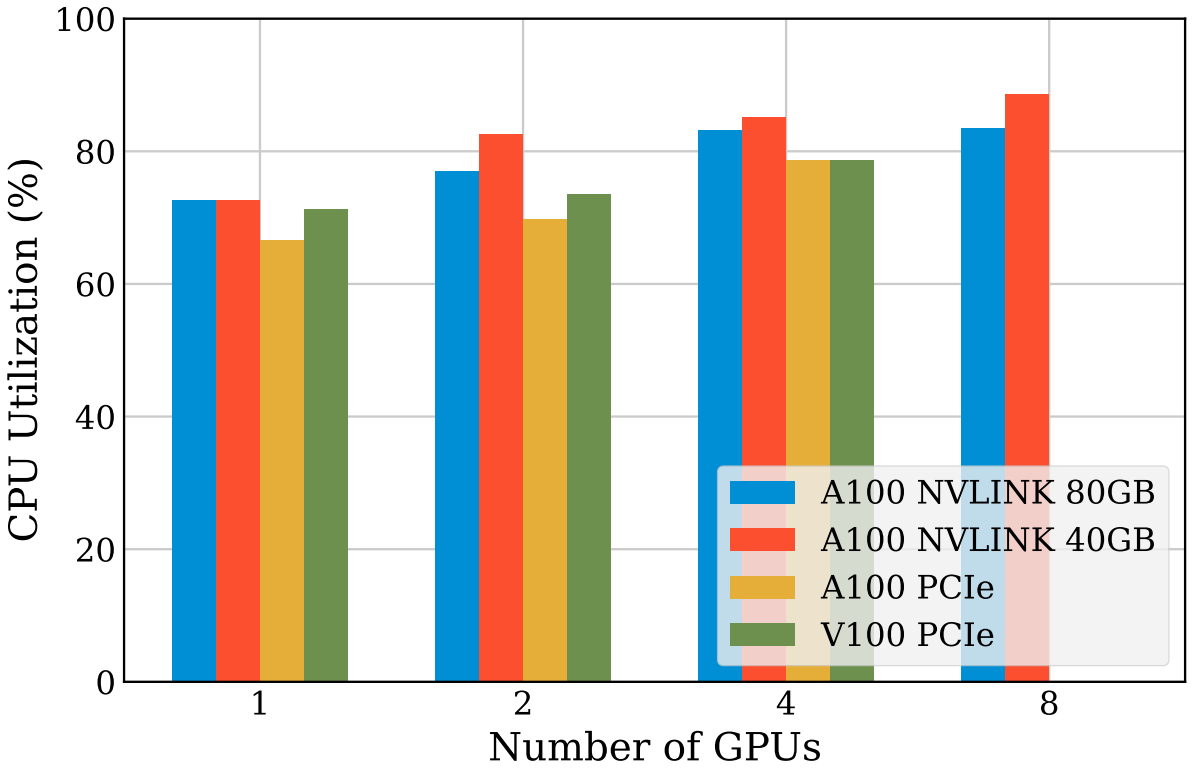

Figure 5: CPU Utilization versus number of GPUs used for Swin Transformer V2 distributed data-parallel training. Metrics gathered via PyTorch Profiler. Error is negligible compared to averages.

5.4 Platform Comparison

A key piece of this work is the performance and scalability comparison of the application of Swin Transformer V2 distributed data-parallel training across multiple hardware platforms. To add emphasis to this analysis, platform comparison figures are made containing speedup, parallel efficiency, GPU utilization, and CPU utilization metrics for the platforms of interest.

The platform comparison chart for CRC A100_NVLINK and CRC A100 nodes is shown in Figure 6. While it may seem as if the NVIDIA A100 SXM is outperformed by the NVIDIA A100 PCIe when looking at each metric, it is important to note that the NVIDIA A100 SXM significantly outperforms the NVIDIA A100 PCIe for single GPU training time by 39% on the NVIDIA A100 SXM 80 GB and 34% on the NVIDIA A100 SXM 40 GB. This difference is of importance as it appears to inflate the speedup and parallel efficiency metrics for the CRC A100 compute node.

For the platform comparison of CRC A100 and CRC V100 nodes, see Figure 7. This platform comparison is used to demonstrate the impact different GPU generations have on performance. Even with the same GPU interconnect (PCIe), CRC A100 outperforms CRC V100 by an average of 24% in training time. A significant reduction in training time can be contributed to the difference in platform generations from the NVIDIA V100 to the newer NVIDIA A100. The A100 has substantially more CUDA cores, updated architecture, newer versions of tensor cores, and interconnect and memory advancements.

Figure 8 contains the platform comparison chart for CRC A100_NVLINK for NVIDIA A100 SXM 80 GB and 40 GB nodes. With CRC A100_NVLINK having 8 GPUs per node, the x-axis on each plot extends to 8 GPUs. The CRC A100_NVLINK nodes with A100 SXM 80 GB GPUs outperform the nodes with A100 SXM 40 GB GPUs by an average of 4%. While seemingly small, this difference is consequential due to the length of each training instance. The 1.3× higher memory bandwidth and 1.35× faster memory clock on the A100

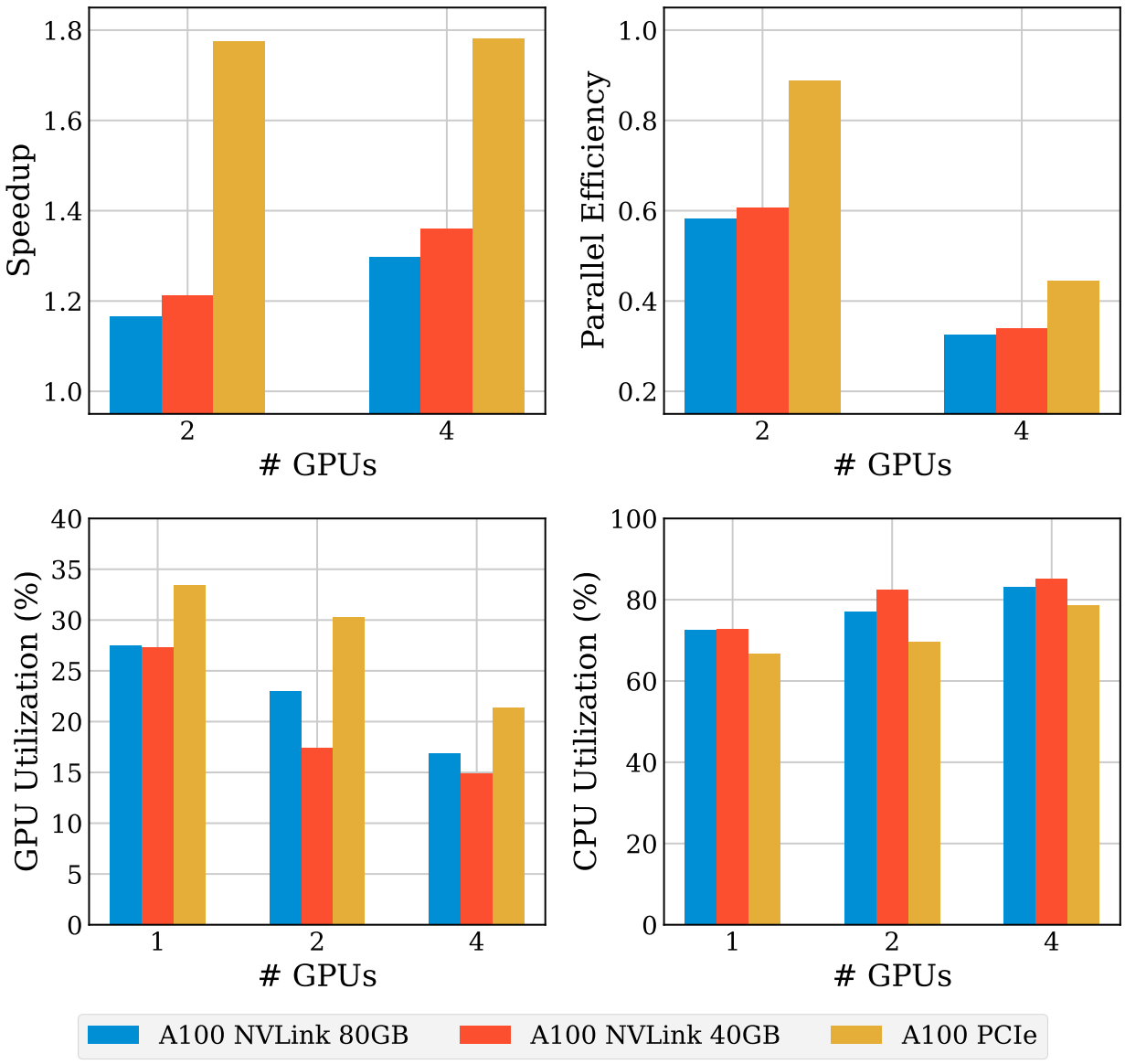

Figure 6: Platform comparison chart for training on CRC A100_NVLINK and CRC A100 nodes. Speedup, parallel efficiency, GPU utilization percentage, and CPU utilization percentage are shown.

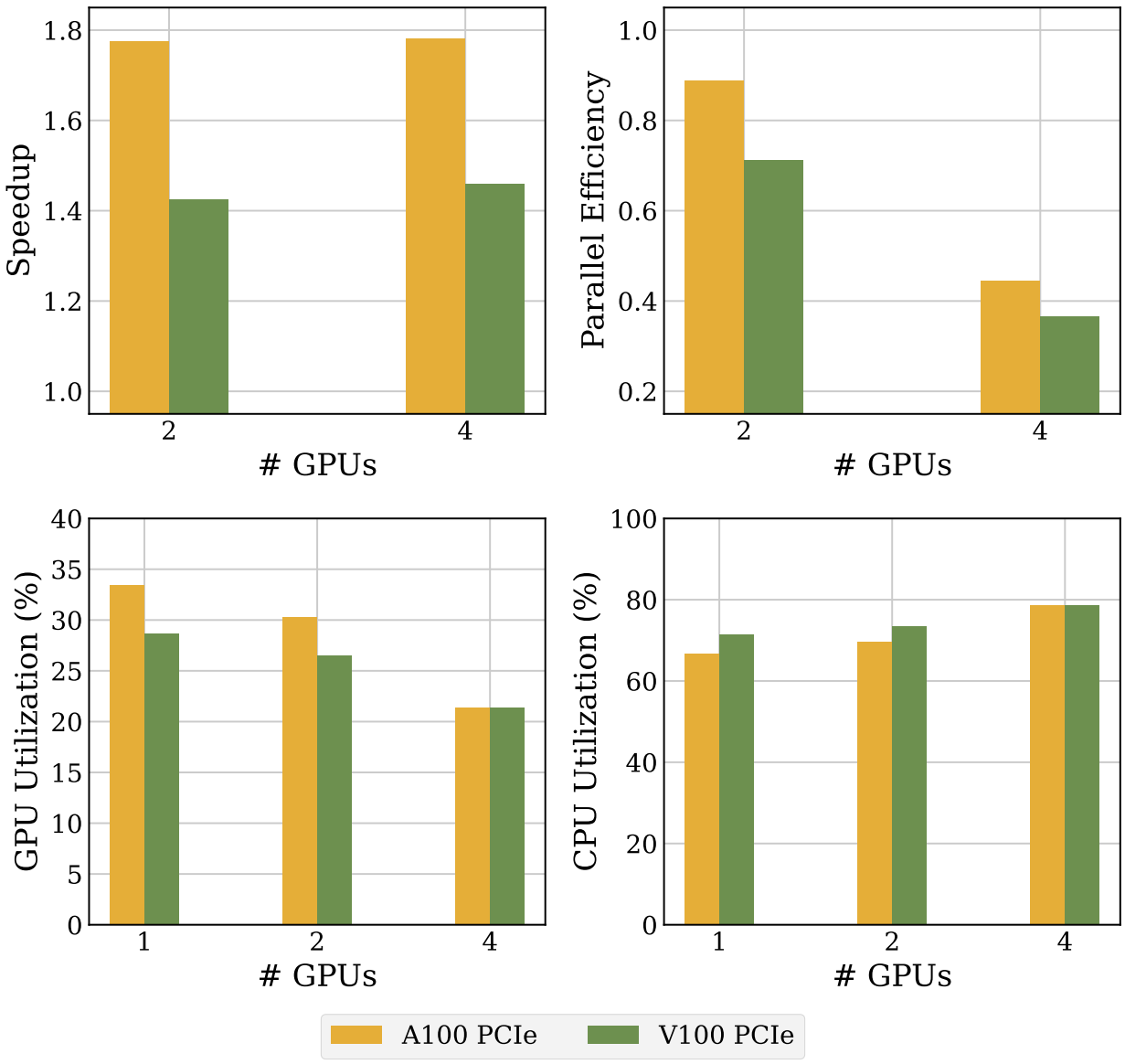

Figure 7: Platform comparison chart for training on CRC A100 and CRC V100 nodes. Speedup, parallel efficiency, GPU utilization percentage, and CPU utilization percentage are displayed.

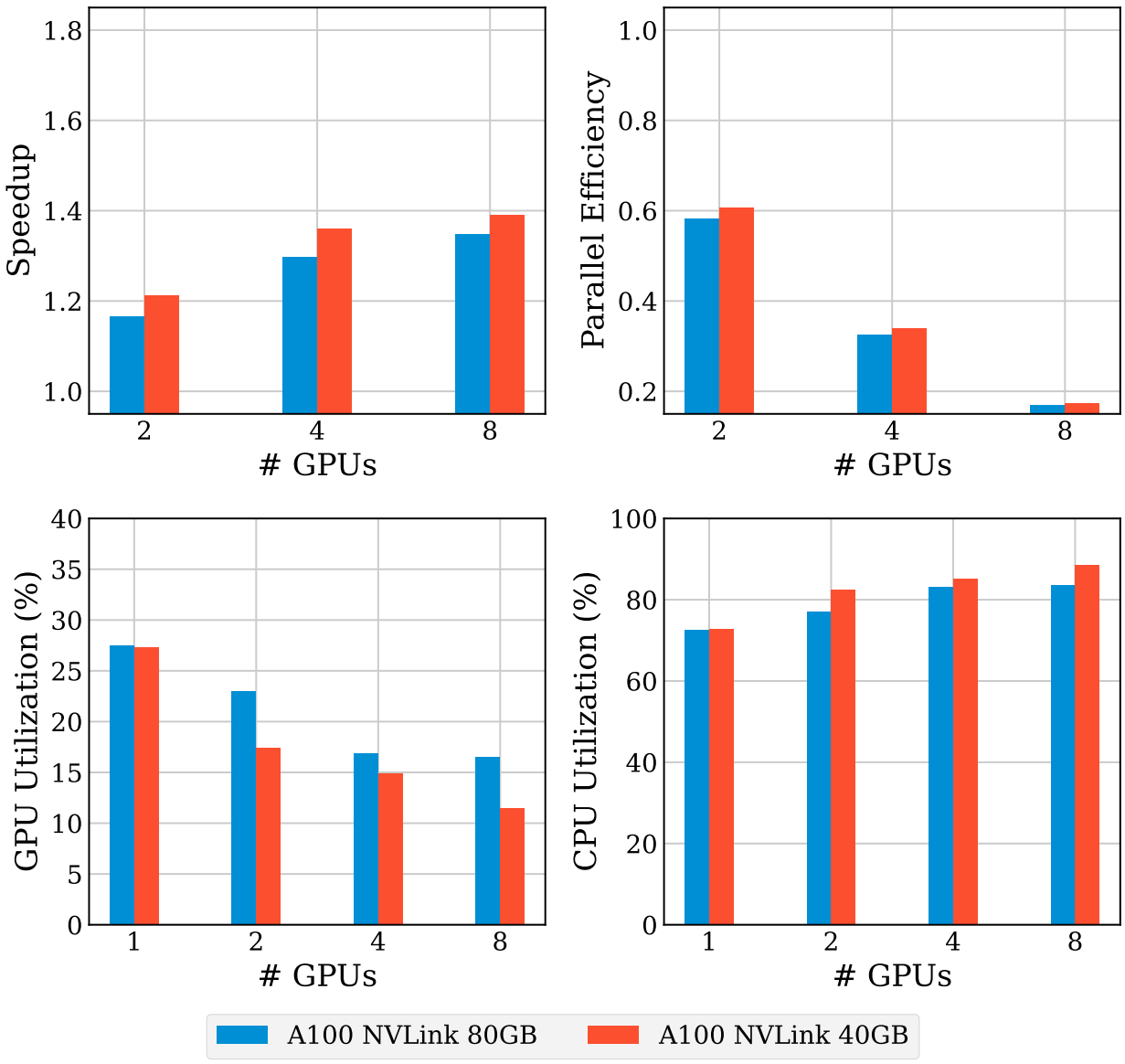

Figure 8: Platform comparison chart for training on CRC A100_NVLINK 80 and 40 GB nodes. Speedup, parallel efficiency, GPU utilization percentage, and CPU utilization percentage are presented.

80 GB card induce faster memory transfers and faster training times overall. However, with a significantly larger dataset that utilizes the full 40 GB of onboard memory for the A100 SXM, one might see a more significant difference in performance between these two platforms.

6.0 Conclusion

This research provides a scalability analysis of the distributed data-parallel training of Swin Transformer V2. Swin Transformer is the first transformer-based vision model to achieve state-of-the-art accuracy for semantic segmentation on the ADE20K dataset. This model is chosen due to an interest in the training scalability for significantly large vision models. To analyze the scalability of this model, a weak scaling experiment is performed by keeping the problem size constant while increasing the number of distributed GPUs used to accelerate training. NVIDIA V100 and NVIDIA A100 GPU nodes with PCIe and NVLINK interconnects hosted by the Center for Research Computing at the University of Pittsburgh are utilized for computing platforms.

The distributed data-parallel training application realizes a significant training time reduction at a small device count. This noteworthy drop is seen as the number of GPUs is increased from 1 to 2, where training time reduces by 26% on average. However, diminishing returns are observed as scaling continues. The change in training time drops to an average of 2.9% when scaling from 4 to 8 GPUs on NVLINK nodes, and drops to an average of 1.3% on PCIe nodes when scaling from 2 to 4 GPUs. As diminishing returns are observed at each scaling step, the claim can be made that distributed data-parallel training of Swin Transformer V2 on ADE20K scales poorly as the number of devices is increased.

As for platform comparisons, the most notable difference in single GPU training time among similar architectures was observed between the NVIDIA A100 PCIe and NVIDIA A100 NVLINK nodes. Single GPU training instances were 39% faster for the NVIDIA A100 NVLINK (80 GB) platform and 34% faster for the NVIDIA A100 NVLINK (40 GB) platform when compared to NVIDIA A100 PCIe. As the GPU interconnects are heavily utilized during data sharing and gradient synchronization, this factor can be contributed to the differences between the NVLINK and PCIe interconnects. Specifically, it can be contributed to the

significantly faster data transfer rate offered by NVLINK. For all experiment configurations, the NVIDIA A100 PCIe node trains 24% faster on average than the NVIDIA V100 PCIe node, demonstrating the benefits of more CUDA cores, updated GPU architecture, and a newer tensor core generation.

7.0 Future Research

This research is limited by the computing platforms, the model sizes trained on, and the data. There is potential to expand from single-node to multi-node distributed GPU platforms for further scalability analysis. Multi-node training offers the ability to analyze the communication overhead introduced with inter-node communication and the potential to analyze asynchronous data-parallel training. As large-model training is of interest, training Swin Transformer V2 models with larger parameter counts should be performed. This would further stress the memory and compute requirements for the DDP algorithm. With at least 32 GB of on-chip memory on each GPU tested in this work, a larger model could potentially offer the ability to train using model parallelism if a single model instance cannot fit on one GPU. Lastly, varying the data mini-batch size (images per GPU per iteration) and training on a larger dataset should be explored. Varying mini-batch sizes will alter the memory and communication requirements for training, and the impact it has on training performance would be of interest. Using a larger dataset to train on would further stress the compute and memory requirements for this application.

8.0 Acknowledgements

This research was supported by the NSF Center for Space, High-performance, and Resilient Computing (SHREC) industry and agency members and by the IUCRC Program of the National Science Foundation under Grant No. CNS-1738783. This research was supported in part by the University of Pittsburgh Center for Research Computing, RRID:SCR_022735, through the resources provided. Specifically, this work used the H2P cluster, which is supported by NSF award number OAC-2117681. I would like to thank my fellow students within the NSF SHREC lab for their help and advice. Specifically, I would like to thank Calvin Gealy, Luke Kljucaric, and Marika Schubert for their guidance in this research. Finally, I would like to thank my friends and family for supporting and encouraging my research and educational endeavors.

Bibliography

[1] Hangbo Bao, Li Dong, and Furu Wei. Beit: Bert pre-training of image transformers. *ArXiv*, abs/2106.08254, 2021.

[2] MMCV Contributors. MMCV: OpenMMLab computer vision foundation. `https://github.com/open-mmlab/mmcv`, 2018.

[3] MMEngine Contributors. MMEngine: Openmmlab foundational library for training deep learning models. 2022.

[4] MMSegmentation Contributors. MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. `https://github.com/open-mmlab/mmsegmentation`, 2020.

[5] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223, 2016.

[6] NVIDIA Corporation. Nvidia a100 tensor core gpu datasheet, 2023.

[7] NVIDIA Corporation. Nvidia v100 tensor core gpu datasheet, 2023.

[8] 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, 2009.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929, 2020.

[11] Paszke et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. pages 8024–8035. Curran Associates, Inc., 2019.

[12] Tom B. Brown et al. Language models are few-shot learners, 2020.

[13] Center for Research Computing at the University of Pittsburgh. Cluster hardware overview.

[14] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32:1231 – 1237, 2013.

[15] Ross B. Girshick. Fast r-cnn. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[16] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Doll'ar, and Ross B. Girshick. Masked autoencoders are scalable vision learners. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15979–15988, 2021.

[17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2018.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[19] Intel. What are pcie 4.0 and 5.0?, 2023.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, may 2017.

[21] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training, 2020.

[22] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin transformer v2: Scaling up capacity and resolution. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11999–12009, 2021.

[23] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9992–10002, 2021.

[24] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2015.

[25] NVIDIA. Nvlink and nvswitch for advanced multi-gpu communication, 2023.

[26] PyTorch. pytorch/pytorch on docker hub.

[27] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[28] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[29] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[30] Vivien Sainte Fare Garnot and Loic Landrieu. Panoptic segmentation of satellite image time series with convolutional temporal attention networks. *ICCV*, 2021.

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[32] Ph.D. Spyridon Bakas. Miccai brats - the multimodal brain tumor segmentation challenge, 2021.

[33] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[34] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[35] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herv'e J'egou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, 2020.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[37] Pablo Villalobos, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, Anson Ho, and Marius Hobbhahn. Machine learning model sizes and the parameter gap, 2022.

[38] Hao Zhang, Feng Li, Siyi Liu, Lei Zhang, Hang Su, Jun-Juan Zhu, Lionel Ming shuan Ni, and Heung yeung Shum. Dino: Detr with improved denoising anchor boxes for end-to-end object detection. *ArXiv*, abs/2203.03605, 2022.

[39] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.