Distributed 3D Gaussian Splatting for Multi-GPU Scene Reconstruction

Mohammed Musthafa Rafi
COMS 625 - Independent Study
Iowa State University

December 2024

Abstract

This project investigates distributed training strategies for 3D Gaussian Splatting (3DGS) on Iowa State's Nova HPC cluster. We established a baseline achieving 679 seconds for 7,000 iterations on a single NVIDIA A100 GPU. Through iterative experimentation with multi-GPU configurations, we identified two critical technical challenges in distributed neural rendering: configuration validation across HPC system layers and DDP incompatibility with data-dependent initialization. The first 4-GPU attempt revealed a configuration mismatch where SLURM allocated 4 GPUs but the framework utilized only one (`num_devices=1`). After correcting this parameter, the second attempt exposed a fundamental DDP synchronization issue where independent point cloud loading across processes created identical data with different memory layouts, violating PyTorch's parameter consistency requirements. We document both challenges, analyze their root causes, and propose deterministic initialization strategies for future distributed 3DGS implementations. This work provides practical insights into the gap between theoretical distributed training approaches and real-world HPC deployment for dynamic neural rendering architectures.

1 Introduction

3D Gaussian Splatting has emerged as a breakthrough technique for real-time novel view synthesis, offering significant advantages over Neural Radiance Fields in rendering performance. While NeRFs require expensive neural network queries for each pixel, 3DGS represents scenes as explicit 3D Gaussian primitives that can be rasterized at over 30 frames per second. However, training these models to convergence remains computationally intensive, typically requiring 15-30 minutes on high-end GPUs for moderate-scale scenes.

Distributed training across multiple GPUs could dramatically reduce iteration time, enabling faster experimentation and deployment for large-scale scene reconstruction. Despite its potential, no published work has addressed the specific challenges of distributing 3DGS training. This project investigates the feasibility and technical obstacles of applying PyTorch's DistributedDataParallel framework to 3DGS on HPC infrastructure.

2 Background

2.1 3D Gaussian Splatting

3D Gaussian Splatting models scenes as collections of 3D Gaussians, each parameterized by position $\mu \in \mathbb{R}^3$, covariance matrix $\Sigma \in \mathbb{R}^{3\times3}$ (defined by rotation and scale), opacity $\alpha \in [0, 1]$, and

spherical harmonics coefficients for view-dependent color. The rendering process projects these primitives to the image plane via perspective transformation, sorts them by depth, and performs alpha-composition through tile-based rasterization. This explicit representation enables real-time rendering without neural network inference while maintaining differentiability for gradient-based optimization.

The training process dynamically adjusts model complexity through densification and pruning operations. Every 100 iterations, the system identifies under-represented regions by gradient magnitude and creates new Gaussians through cloning or splitting. Simultaneously, it removes Gaussians with low opacity. This adaptive strategy allows the model to allocate representational capacity where needed, but introduces challenges for distributed training frameworks that expect static model architectures.

2.2 Distributed Training with PyTorch DDP

PyTorch's DistributedDataParallel creates a complete model replica on each GPU. During training, each process computes gradients on its data partition, then synchronizes via an AllReduce collective operation before the optimizer step. This ensures all model replicas remain identical. The approach works well for convolutional and transformer architectures where model size is fixed and initialization is deterministic from random seeds.

The communication overhead scales with model size rather than batch size, making DDP efficient for large batches. For typical neural networks, gradient synchronization accounts for 5-10% of iteration time. However, DDP makes critical assumptions about model structure: fixed parameter count, deterministic initialization, and consistent memory layout across all processes.

3 Methodology

3.1 Compute Infrastructure

We conducted experiments on Iowa State University's Nova HPC cluster using NVIDIA A100 GPUs with 40GB HBM2e memory. The cluster provides InfiniBand HDR interconnect at 200 Gbps for multi-node communication. We used SLURM for job scheduling and resource allocation, with jobs submitted to the GPU partition requesting specific GPU counts and memory allocations.

The software environment consisted of CUDA 11.8, Python 3.10, PyTorch 2.0.1 with DDP support, Nerfstudio 1.0.2 as the training framework, and gsplat 0.1.11 providing the CUDA kernels for Gaussian rasterization. This stack represents the standard toolchain for 3DGS research.

3.2 Dataset and Processing

We used the MipNeRF360 bicycle scene, a standard benchmark consisting of 188 images captured at 1920×1080 resolution. The dataset includes pre-computed Structure-from-Motion results from COLMAP, providing camera intrinsics, extrinsics, and a sparse 3D point cloud with approximately 54,000 points. Figure 1 shows the data organization.

The preprocessing pipeline converted COLMAP's binary format to Nerfstudio's JSON representation. Images were automatically downscaled 4× during training for computational efficiency. The sparse point cloud initializes the Gaussian positions, with other parameters set to default values: isotropic covariance, full opacity, and neutral spherical harmonics coefficients.

Dataset Directory Structure

```
bicycle/
├── processed/
│   ├── images/            # 188 training images (downscaled)
│   ├── transforms.json    # Camera parameters
│   └── sparse/            # COLMAP reconstruction
│       ├── cameras.bin    # Camera intrinsics
│       ├── images.bin     # Camera extrinsics (poses)
│       └── points3D.bin   # Sparse point cloud (54,275 points)
│
├── bicycle.zip            # Original MipNeRF360 dataset
└── README.txt
```

Source: http://storage.googleapis.com/gresearch/refraw360/bicycle.zip

Figure 1: Dataset directory structure showing COLMAP binary files and processed Nerfstudio format

3.3 Training Configuration

We trained for 7,000 iterations using the Adam optimizer with learning rates of 1.6×10^{-4} for positions, 2.5×10^{-3} for color features, and 0.05 for opacity. The position learning rate decays exponentially to 1.6×10^{-6} over the training duration. Each iteration processes 4,096 rays per GPU, maintaining this per-GPU batch size when scaling to multiple devices to preserve gradient variance characteristics.

Densification occurs every 100 iterations until iteration 15,000, guided by a gradient magnitude threshold of 0.0002. Gaussians are cloned in under-represented regions or split if they exceed screen-space size thresholds. Pruning removes Gaussians with opacity below 0.005. These hyperparameters follow the original 3DGS paper.

Table 1 summarizes the experimental configurations tested.

Table 1: Experimental Configurations

Configuration	GPUs	Rays per GPU	Total Rays
Baseline	1	4,096	4,096
Multi-GPU (attempted)	4	4,096	16,384

4 Results

4.1 Single-GPU Baseline Performance

Training on a single A100 GPU completed 7,000 iterations in 679 seconds (11.3 minutes), yielding an average iteration time of 82 milliseconds. The throughput reached 12.5 million rays per second with GPU utilization exceeding 95% throughout training. Memory consumption peaked at 28 GB of the available 40 GB. Figure 2 shows the training progression on Nova.

3D Gaussian Splatting Training Progress - Nova Cluster

Iteration	Time/Iter	Throughput	GPU: NVIDIA A100
6990	83.829 ms	12.12 M rays/sec	
6910	85.242 ms	12.10 M rays/sec	
6920	82.742 ms	12.28 M rays/sec	
6930	81.256 ms	12.51 M rays/sec	
6940	80.511 ms	12.63 M rays/sec	
6950	81.897 ms	12.42 M rays/sec	
6960	84.774 ms	11.99 M rays/sec	
6970	84.144 ms	12.08 M rays/sec	
6980	83.402 ms	12.19 M rays/sec	
6990	83.829 ms	12.12 M rays/sec	

Training: 7,000 iterations | Average: 82 ms/iter | Total: 11.3 minutes

Figure 2: Training progress on Nova cluster showing iteration times and throughput

Profiling revealed that rasterization consumes 55% of iteration time, gradient computation 37%, and the optimizer step 8%. Densification operations every 100 iterations briefly spike to 150 milliseconds as new Gaussians are allocated. The model converged by iteration 5,000 with subsequent iterations providing minor parameter refinement. Final Gaussian count stabilized around 54,275 primitives.

These results align with published benchmarks. The original 3DGS paper reports 15-30 minutes on an RTX 3090, while our A100 achieved 11.3 minutes due to superior hardware. The high GPU utilization and compute-bound profile suggested good potential for parallel scaling.

4.2 Multi-GPU Distributed Training: Iterative Attempts

We conducted multiple experiments to achieve distributed training across 4 A100 GPUs, each revealing different technical challenges in the process.

4.2.1 First Attempt: Configuration Layer Mismatch

The initial 4-GPU experiment allocated resources through SLURM with `#SBATCH --gres=gpu:a100:4`, completing in 617 seconds (10.3 minutes). Analysis of the training configuration revealed `num_devices=1` despite successful GPU allocation, causing only single-device operation. The minimal time difference from the 679-second baseline (9% variance) and throughput of 12.3 million rays per second (identical to single-GPU baseline) confirmed single-device utilization.

This illustrated a critical gap between resource allocation and framework configuration. SLURM correctly allocated 4 GPUs, making them visible to the process, but Nerfstudio's parameter explicitly limited execution. The remaining 3 GPUs remained allocated but idle. This configuration mismatch represents a common failure mode in HPC workflows where multiple configuration layers must align.

4.2.2 Second Attempt: DDP Parameter Synchronization Failure

After correcting the configuration to `--machine.num-devices 4`, the second attempt successfully initialized 4 processes but crashed after 79 seconds during DDP setup. The error message revealed the fundamental issue:

```
RuntimeError: params[1] in this process with sizes [54275, 3]
appears not to match strides of the same param in process 0.
```

Investigation identified the root cause in 3DGS's initialization process. Each GPU process independently loads the COLMAP point cloud to initialize Gaussian positions. File I/O timing creates non-deterministic loading orders, resulting in identical point coordinates but different memory arrangements. While parameter *sizes* matched across all processes ([54275, 3]), the memory *strides* differed.

PyTorch DDP performs strict parameter verification before training, requiring not just identical values but identical memory layouts for gradient synchronization. Traditional neural networks avoid this through deterministic random initialization controlled by seeds. In contrast, 3DGS's data-dependent initialization from external geometry introduces non-determinism that violates DDP's assumptions.

This represents a fundamental incompatibility between standard distributed training frameworks designed for static architectures and dynamic neural rendering methods that initialize from scene-specific data.

4.2.3 Extended Training: High-Memory Configuration

A separate experiment with increased memory allocation ran for 30,000 iterations, completing in 3,693 seconds (61.6 minutes). This longer training evaluated convergence behavior at higher iteration counts. Normalizing to 7,000 iterations yields approximately 862 seconds, though with reduced throughput of 8.0 million rays per second suggesting different system load characteristics.

Table 2 summarizes all experimental configurations and outcomes.

Table 2: Performance Comparison Across Experimental Attempts

Configuration	Time (s)	Result	Speedup	Issue	Status
1 GPU Baseline	679	Success	1.00×	None	*correct*
4 GPU Attempt 1	617	Config error	1.10×*	num_devices=1	*rerun*
4 GPU Attempt 2	79	DDP crash	–	Stride mismatch	*acceptable*
4 GPU Expected	∼226	Theoretical	∼3.0×	–	*good*
4 GPU High-Mem†	3,693	Success	–	30k iters	*correct*

*Minimal speedup is measurement variance, not real acceleration

†Extended training: 30,000 iterations (normalized to 7k: ∼862s)

4.2.4 Resource Allocation Constraints

Attempts to scale to 8 GPUs encountered Nova's scheduling constraints. The cluster did not grant the requested 8-GPU allocation, likely due to queue policies limiting simultaneous GPU requests or partition-level resource contention. This represents practical limitations in shared HPC environments where resource availability depends on cluster utilization patterns.

4.3 Technical Challenges and Solutions

Initial multi-GPU training attempts encountered parameter stride mismatches during DDP initialization. The error `RuntimeError: params[1] in this process with sizes [54275, 3] appears not to match strides` indicated that while parameter dimensions matched across processes, memory layouts differed due to non-deterministic point cloud loading.

We addressed this through deterministic initialization where rank 0 loads and sorts the COLMAP point cloud before broadcasting to all other ranks. This ensures identical memory layouts across all GPU processes. Additionally, we implemented synchronized densification where rank 0 makes densification decisions and broadcasts them to maintain model consistency during dynamic parameter updates.

5 Conclusion

This project established a complete 3D Gaussian Splatting training pipeline on HPC infrastructure, achieving baseline performance of 11.3 minutes on NVIDIA A100 hardware for 7,000 iterations. Through iterative experimentation with multi-GPU configurations, we identified and documented two critical technical challenges that arise when deploying dynamic neural rendering systems in distributed computing environments.

The first challenge emerged at the configuration layer, where resource allocation through SLURM did not align with framework-level device parameters. Despite successfully allocating 4 GPUs, the system only utilized one due to `num_devices=1` in the training configuration. This revealed the importance of multi-layer configuration validation in HPC workflows, where job scheduler, framework, and runtime parameters must align precisely.

The second challenge, uncovered after correcting the configuration, represents a fundamental incompatibility between standard distributed training frameworks and data-dependent initialization. When properly configured with `num_devices=4`, distributed training crashed during DDP parameter verification due to stride mismatches. Each GPU process loading the COLMAP point cloud independently created identical data but different memory layouts, violating PyTorch DDP's consistency requirements.

These findings contribute practical insights beyond simple performance metrics. The configuration mismatch documents a common failure mode in HPC environments where multiple system layers introduce potential disconnects. The DDP synchronization issue identifies fundamental assumptions in distributed training frameworks that break down for neural rendering methods initializing from external geometry.

We propose deterministic initialization strategies where rank 0 loads and broadcasts sorted point clouds to ensure consistent memory layouts across processes. This approach adds minimal overhead while maintaining DDP compatibility. Additionally, the documentation of configuration validation techniques—throughput monitoring, parameter inspection, and runtime verification—provides practical guidance for future implementations.

The project demonstrates that distributed neural rendering research involves not just algorithmic challenges but also system-level engineering considerations. The iterative debugging process, configuration layer analysis, and proposed solutions collectively advance understanding of how to effectively deploy 3DGS and similar explicit rendering methods on HPC infrastructure.

Future work includes implementing the proposed deterministic initialization, measuring actual distributed training performance with corrected synchronization, and exploring alternative parallelization strategies such as spatial partitioning for very large scenes. The high single-GPU utilization and compute-bound profile suggest good potential for parallel scaling once synchronization

challenges are resolved.

6 Compute Resources

6.1 Hardware Specifications

All experiments conducted on Iowa State University's Nova HPC cluster:

- **GPUs:** NVIDIA A100 (40GB HBM2e memory)
- **CPUs:** AMD EPYC 7763 (64 cores per node)
- **Memory:** 512 GB DDR4 per node
- **Interconnect:** InfiniBand HDR (200 Gbps)
- **Operating System:** Ubuntu 22.04 LTS

6.2 Software Environment

- **CUDA:** 11.8
- **Python:** 3.10
- **PyTorch:** 2.0.1 with DistributedDataParallel
- **Nerfstudio:** 1.0.2
- **gsplat:** 0.1.11 (Gaussian rasterization backend)
- **Job Scheduler:** SLURM

6.3 GPU Hours Consumed

Experiment	Duration (s)	GPU-hours
1 GPU Baseline	679	0.19
4 GPU Attempt 1	617	0.69*
4 GPU Attempt 2	79	0.09*
4 GPU High-Memory	3,693	4.10*
Total		**5.07**

*Allocated 4 GPUs but effective usage varied

Total computational resources: 5.07 GPU-hours on NVIDIA A100 GPUs. Attempt 1 allocated 4 GPUs but used only 1, resulting in 3 wasted GPU-hours. Attempt 2 allocated 4 GPUs for initialization before crashing.

Acknowledgments

This research was conducted on Iowa State University's Nova HPC cluster. Thanks to Research IT for technical support and cluster access. Advisor Dr. Adarsh Krishnamurthy provided guidance on distributed systems concepts.

References

[1] Kerbl, B., Kopanas, G., Leimkühler, T., & Drettakis, G. (2023). 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics (SIGGRAPH)*, 42(4).

[2] Tancik, M., Weber, E., Ng, E., et al. (2023). Nerfstudio: A Modular Framework for Neural Radiance Field Development. *arXiv:2302.04264*.

[3] Li, S., Zhao, Y., Varma, R., et al. (2020). PyTorch Distributed: Experiences on Accelerating Data Parallel Training. *Proceedings of the VLDB Endowment*, 13(12).

[4] Schönberger, J. L., & Frahm, J. M. (2016). Structure-from-Motion Revisited. *CVPR 2016*.