DistriFusion: Distributed Parallel Inference for High-Resolution Diffusion Models

Muyang Li[1*] Tianle Cai[2*] Jiaxin Cao[3] Qinsheng Zhang[4] Han Cai[1]
Junjie Bai[3] Yangqing Jia[3] Kai Li[2] Song Han[1,4]

[1]MIT [2]Princeton [3]Lepton AI [4]NVIDIA
https://github.com/mit-han-lab/distrifuser

Prompt: *Ethereal fantasy concept art of an elf, magnificent, celestial, ethereal, painterly, epic, majestic, magical, fantasy art, cover art, dreamy.*

Figure 1. We introduce DistriFusion, a training-free algorithm to harness multiple GPUs to accelerate diffusion model inference without sacrificing image quality. Naïve Patch (Figure 2(b)) suffers from the fragmentation issue due to the lack of patch interaction. Our DistriFusion removes artifacts and avoids the communication overhead by reusing the features from the previous steps. Setting: SDXL with 50-step Euler sampler, 1280 × 1920 resolution. Latency is measured on A100s.

Abstract

Diffusion models have achieved great success in synthesizing high-quality images. However, generating high-resolution images with diffusion models is still challenging due to the enormous computational costs, resulting in a prohibitive latency for interactive applications. In this paper, we propose DistriFusion to tackle this problem by leveraging parallelism across multiple GPUs. Our method splits the model input into multiple patches and assigns each patch to a GPU. However, naïvely implementing such an algorithm breaks the interaction between patches and loses fidelity, while incorporating such an interaction will incur tremendous communication overhead. To overcome this dilemma, we observe the high similarity between the input from adjacent diffusion steps and propose displaced patch parallelism, which takes advantage of the sequential nature of the diffusion process by reusing the pre-computed feature maps from the previous timestep to provide context for the current step. Therefore, our method supports asynchronous communication, which can be pipelined by computation. Extensive experiments show that our method can be applied to recent Stable Diffusion XL with no quality degradation and achieve up to a 6.1× speedup on eight A100 GPUs compared to one.

*indicates equal contributions.

1. Introduction

The advent of AI-generated content (AIGC) represents a seismic shift in technological innovation. Tools like Adobe Firefly, Midjourney and recent Sora showcase astonishing capabilities, producing compelling imagery and designs from simple text prompts. These achievements are notably supported by the progression in diffusion models [13, 57]. The emergence of large text-to-image models, including Stable Diffusion [51], Imgen [53], eDiff-I [2], DALL·E [3, 45, 46] and Emu [6], further expands the horizons of AI creativity. Trained on diverse open-web data, these models can generate photorealistic images from text descriptions alone. Such technological revolution unlocks numerous synthesis and editing applications for images and videos, placing new demands on responsiveness: by *interactively* guiding and refining the model output, users can achieve more personalized and precise results. Nonetheless, a critical challenge remains – high resolution leading to large computation. For example, the original Stable Diffusion [51] is limited to generating 512 × 512 images. Later, SDXL [43] expands the capabilities to 1024 × 1024 images. More recently, Sora further pushes the boundaries by enabling video generation at 1080 × 1920 resolution. Despite these advancements, the increased latency of generating high-resolution images

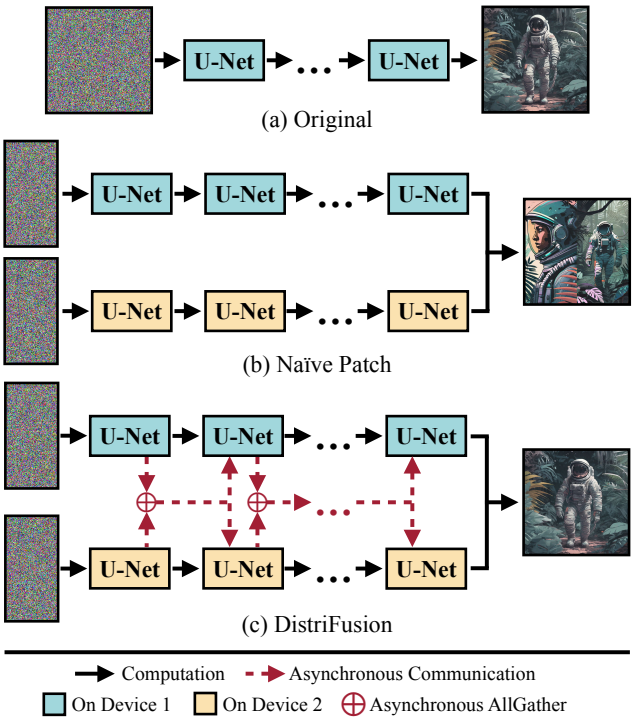

Figure 2. (a) Original diffusion model running on a single device. (b) Naïvely splitting the image into 2 patches across 2 GPUs has an evident seam at the boundary due to the absence of interaction across patches. (c) DistriFusion employs synchronous communication for patch interaction at the first step. After that, we reuse the activations from the previous step via asynchronous communication. In this way, the communication overhead can be hidden into the computation pipeline.

presents a tremendous barrier to real-time applications.

Recent efforts to accelerate diffusion model inference have mainly focused on two approaches: reducing sampling steps [20, 32–34, 54, 58, 66, 69] and optimizing neural network inference [23, 25, 26]. As computational resources grow rapidly, leveraging multiple GPUs to speed up inference is appealing. For example, in natural language processing (NLP), large language models have successfully harnessed tensor parallelism across GPUs, significantly reducing latency. However, for diffusion models, multiple GPUs are usually only used for batch inference. When generating a single image, typically only one GPU is involved (Figure 2(a)). Techniques like tensor parallelism are less suitable for diffusion models due to the large activation size, as communication costs outweigh savings from distributed computation. Thus, even when multiple GPUs are available, they cannot be effectively exploited to further accelerate single-image generation. This motivates the development of a method that can utilize multiple GPUs to speed up single-image generation with diffusion models.

A naïve approach would be to divide the image into several patches, assigning each patch to a different device for generation, as illustrated in Figure 2(b). This method allows for independent and parallel operations across devices. However, it suffers from a clearly visible seam at the boundaries of each patch due to the absence of interaction between the individual patches. However, introducing interactions among patches to address this issue would incur excessive synchronization costs again, offsetting the benefits of parallel processing.

In this work, we present *DistriFusion*, a method that enables running diffusion models across multiple devices in parallel to reduce the latency of single-sample generation without hurting image quality. As depicted in Figure 2(c), our approach is also based on patch parallelism, which divides the image into multiple patches, each assigned to a different device. Our key observation is that the inputs across adjacent denoising steps in diffusion models are similar. Therefore, we adopt synchronous communication solely for the first step. For the subsequent steps, we reuse the pre-computed activations from the *previous step* to provide global context and patch interactions for the *current step*. We further co-design an inference framework to implement our algorithm. Specifically, our framework effectively hides the communication overhead within the computation via asynchronous communication. It also sparsely runs the convolutional and attention layers exclusively on the assigned regions, thereby proportionally reducing per-device computation. Our method, distinct from data, tensor, or pipeline parallelism, introduces a new parallelization opportunity: *displaced patch parallelism*.

DistriFusion only requires off-the-shelf pre-trained diffusion models and is applicable to a majority of few-step samplers. We benchmark it on a subset of COCO Captions [5]. Without loss of visual fidelity, it mirrors the performance of the original Stable Diffusion XL (SDXL) [43] while reducing the computation* proportionally to the number of used devices. Furthermore, our framework also reduces the latency of SDXL U-Net for generating a single image by up to 1.8×, 3.4× and 6.1× with 2, 4, and 8 A100 GPUs, respectively. When combined with batch splitting for classifier-free guidance [12], we achieve in total 3.6× and 6.6× speedups using 4 and 8 A100 GPUs for 3840×3840 images, respectively. See Figure 1 for some examples of our method.

2. Related Work

Diffusion models. Diffusion models have significantly transformed the landscape of content generation [2, 13, 39, 43]. At its core, these models synthesize content through an iterative denoising process. Although this iterative approach yields unprecedented capabilities for content generation, it requires substantially more computational resources and

*Following previous works, we measure the computational cost with the number of Multiply-Accumulate operations (MACs). 1 MAC=2 FLOPs.

results in slower generative speed. This issue intensifies with the synthesis of high-dimensional data, such as high-resolution [9, 14] or 360° images [71]. Researchers have investigated various perspectives to accelerate the diffusion model. The first line lies in designing more efficient denoising processes. Rombach *et al*. [51] and Vahdat *et al*. [62] propose to compress high-resolution images into low-resolution latent representations and learn diffusion model in latent space. Another line lies in improving sampling via designing efficient training-free sampling algorithms. A large category of works along this line is built upon the connection between diffusion models and differential equations [59], and leverage a well-established exponential integrator [32, 69, 70] to reduce sampling steps while maintaining numerical accuracy. The third strategy involves distilling faster generative models from pre-trained diffusion models. Despite significant progress made in this area, a quality gap persists between these expedited generators and diffusion models [19, 34, 54]. In addition to the above schemes, some works investigate how to optimize the neural inference for diffusion models [23, 25, 26]. In this work, we explore a new paradigm for accelerating diffusion by leveraging parallelism to the neural network on multiple devices.

Parallelism. Existing work has explored various parallelism strategies to accelerate the training and inference of large language models (LLMs), including data, pipeline [15, 27, 36], tensor [17, 37, 67, 68, 74], and zero-redundancy parallelism [44, 47, 48, 73]. Tensor parallelism, in particular, has been widely adopted for accelerating LLMs [28], which are characterized by their substantial model sizes, whereas their activation sizes are relatively small. In such scenarios, the communication overhead introduced by tensor parallelism is relatively minor compared to the substantial latency benefits brought by increased memory bandwidth. However, the situation differs for diffusion models, which are generally smaller than LLMs but are often bottlenecked by the large activation size due to the spatial dimensions, especially when generating high-resolution content. The communication overhead from tensor parallelism becomes a significant factor, overshadowing the actual computation time. As a result, only data parallelism has been used thus far for diffusion model serving, which provides no latency improvements. The only exception is ParaDiGMS [56], which uses Picard iteration to run multiple steps in parallel. However, this sampler tends to waste much computation, and the generated results exhibit significant deviation from the original diffusion model. Our method is based on patch parallelism, which distributes the computation across multiple devices by splitting the input into small patches. Compared to tensor parallelism, such a scheme has superior independence and reduced communication demands. Additionally, it favors the use of `AllGather` over `AllReduce` for data interaction, significantly lowering overhead (see Section 5.3 for the full comparisons). Drawing inspiration from the success of asynchronous communication in parallel computing [63], we further reuse the features from the previous step as context for current step to overlap communication and computation, called *displaced patch parallelism*. This represents the first parallelism strategy tailored to the sequential characteristics of diffusion models while avoiding the heavy communication costs of traditional techniques like tensor parallelism.

Sparse computation. Sparse computation has been extensively researched in various domains, including weight [10, 16, 21, 31], input [50, 60, 61] and activation [7, 18, 23, 24, 40, 49, 49, 55]. In the activation domain, to facilitate on-hardware speedups, several studies propose to use structured sparsity. SBNet [49] employs a spatial mask to sparsify activations for accelerating 3D object detection. This mask can be derived either from prior problem knowledge or an auxiliary network. In the context of image generation, SIGE [23] leverages the highly structured sparsity of user edits, selectively performing computation at the edited regions to speed up GANs [8] and diffusion models. MCUNetV2[29] adopts a patch-based inference to reduce memory usage for image classification and detection. In our work, we also partition the input into patches, each processed by a different device. However, we focus on reducing the latency by parallelism for image generation instead. Each device will solely process the assigned regions to reduce the per-device computation.

3. Background

To generate a high-quality image, a diffusion model often trains a noise-prediction neural model (*e.g.*, U-Net [52]) ϵ_θ. Starting from pure Gaussian noise $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, it involves tens to hundreds of iterative denoising steps to get the final clean image $\mathbf{x}_0$, where T is the total number of steps. Specifically, given the noisy image $\mathbf{x}_t$ at time step t, the model ϵ_θ takes $\mathbf{x}_t$, t and an additional condition c (*e.g.*, text) as inputs to predict the corresponding noise ϵ_t within $\mathbf{x}_t$. At each denoising step, $\mathbf{x}_{t-1}$ can be derived from the following equation:

$$\mathbf{x}_{t-1} = \text{Update}(\mathbf{x}_t, t, \epsilon_t), \quad \epsilon_t = \epsilon_\theta(\mathbf{x}_t, t, c). \tag{1}$$

Here, 'Update' refers to a sampler-specific function that typically includes element-wise additions and multiplications. Therefore, the primary source of latency in this process is the forward passes through model ϵ_θ. For example, Stable Diffusion XL [43] requires 6,763 GMACs per step to generate a 1024×1024 image. This computational demand escalates more than quadratically with increasing resolution, making the latency for generating a single high-resolution image impractically high for real-world applications. Furthermore, given that $\mathbf{x}_{t-1}$ depends on $\mathbf{x}_t$, parallel computation of ϵ_t and ϵ_{t-1} is challenging. Hence, even with multiple idle GPUs, accelerating the generation of a single high-

resolution image remains tricky. Recently, Shih *et al*. introduced ParaDiGMS [56], employing Picard iterations to parallelize the denoising steps in a data-parallel manner. However, ParaDiGMS wastes the computation on speculative guesses that fail quality thresholds. It also relies on a large total step count T to exploit multi-GPU data parallelism, limiting its potential applications. Another conventional method is sharding the model on multiple devices and using tensor parallelism for inference. However, this method suffers from intolerable communication costs, making it impractical for real-world applications. Beyond these two schemes, are there alternative strategies for distributing workloads across multiple GPU devices so that single-image generation can also enjoy the free-lunch speedups from multiple devices?

4. Method

The key idea of DistriFusion is to parallelize computation across devices by splitting the image into patches. Naïvely, this can be done by either (1) independently computing patches and stitching them together, or (2) synchronously communicating intermediate activations between patches. However, the first approach leads to visible discrepancies at the boundaries of each patch due to the absence of interaction between them (see Figure 1 and Figure 2(b)). The second approach, on the other hand, incurs excessive communication overheads, negating the benefits of parallel processing. To address these challenges, we propose a novel parallelism paradigm, *displaced patch parallelism*, which leverages the sequential nature of diffusion models to overlap communication and computation. Our key insight is reusing slightly outdated, or 'stale' activations from the previous diffusion step to facilitate interactions between patches, which we describe as *activation displacement*. This is based on the observation that the inputs for consecutive denoising steps are relatively similar. Consequently, computing each patch's activation at a layer does not rely on other patches' fresh activations, allowing communication to be hidden within subsequent layers' computation. We will next provide a detailed breakdown of each aspect of our algorithm and system design.

Displaced patch parallelism. As shown in Figure 3, when predicting $\epsilon_\theta(\mathbf{x}_t)$ (we omit the inputs of timestep t and condition c here for simplicity), we first split $\mathbf{x}_t$ into multiple patches $\mathbf{x}_t^{(1)}, \mathbf{x}_t^{(2)}, \ldots, \mathbf{x}_t^{(N)}$, where N is the number of devices. For example, we use $N = 2$ in Figure 3. Each device has a replicate of the model ϵ_θ and will process a single patch independently, in parallel.

For a given layer l, let's consider the input activation patch on the i-th device, denoted as $A_t^{l,(i)}$. This patch is first scattered into the stale activations from the previous step, A_{t+1}^l, at its corresponding spatial location (the method for obtaining A_{t+1}^l will be discussed later). Here, A_{t+1}^l is in full spatial shape. In the `Scatter` output, only the $\frac{1}{N}$ regions where $A_t^{l,(i)}$ is placed are fresh and require recomputation. We then selectively apply the layer operation F_l (linear, convolution, or attention) to these fresh areas, thereby generating the output for the corresponding regions. This process is repeated for each layer. Finally, the outputs from all layers are synchronized together to approximate $\epsilon_\theta(\mathbf{x}_t)$. Through this methodology, each device is responsible for only $\frac{1}{N}$ of the total computations, enabling efficient parallelization.

There still remains a problem of how to obtain the stale activations from the previous step. As shown in Figure 3, at each timestep t, when device i acquires $A_t^{l,(i)}$, it will then broadcast the activations to all other devices and perform the `AllGather` operation. Modern GPUs often support asynchronous communication and computation, which means that this `AllGather` process does not block ongoing computations. By the time we reach layer l in the next timestep, each device should have already received a replicate of A_t^l. Such an approach effectively hides communication overheads within the computation phase, as shown in Figure 4. However, there is an exception: the very first step (*i.e*., $\mathbf{x}_T$). In this scenario, each device simply executes the standard synchronous communication and caches the intermediate activations for the next step.

Sparse operations. For each layer l, we modify the original operator F_l to enable sparse computation selectively on the fresh areas. Specifically, if F_l is a convolution, linear, or cross-attention layer, we apply the operator exclusively to the newly refreshed regions, rather than the full feature map. This can be achieved by extracting the fresh sections from the `scatter` output and feeding them into F_l. For layers where F_l is a self-attention layer, we transform it into a cross-attention layer, similar to SIGE [23]. In this setting, only the query tokens from the fresh areas are preserved on the device, while the key and value tokens still encompass the entire feature map (the `scatter` output). Thus, the computational cost for F_l is exactly proportional to the size of the fresh area.

Corrected asynchronous GroupNorm. Diffusion models often adopt group normalization (GN) [38, 64] layers in the network. These layers normalize across the spatial dimension, necessitating the aggregation of activations to restore their full spatial shape. In Section 5.3, we discover that either normalizing only the fresh patches or reusing stale features degrades image quality. However, aggregating all the normalization statistics will incur considerable overhead due to the synchronous communication. To solve this dilemma, we additionally introduce a correction term to the stale statistics. Specifically, for each device i at a given step t, every GN layer can compute the group-wise mean of its fresh patch $\mathbf{A}_t^{(i)}$, denoted as $\mathbb{E}[\mathbf{A}_t^{(i)}]$. For simplicity, we omit the layer index l here. It also has cached the local mean $\mathbb{E}[\mathbf{A}_{t+1}^{(i)}]$ and aggregated global mean $\mathbb{E}[\mathbf{A}_{t+1}]$ from the

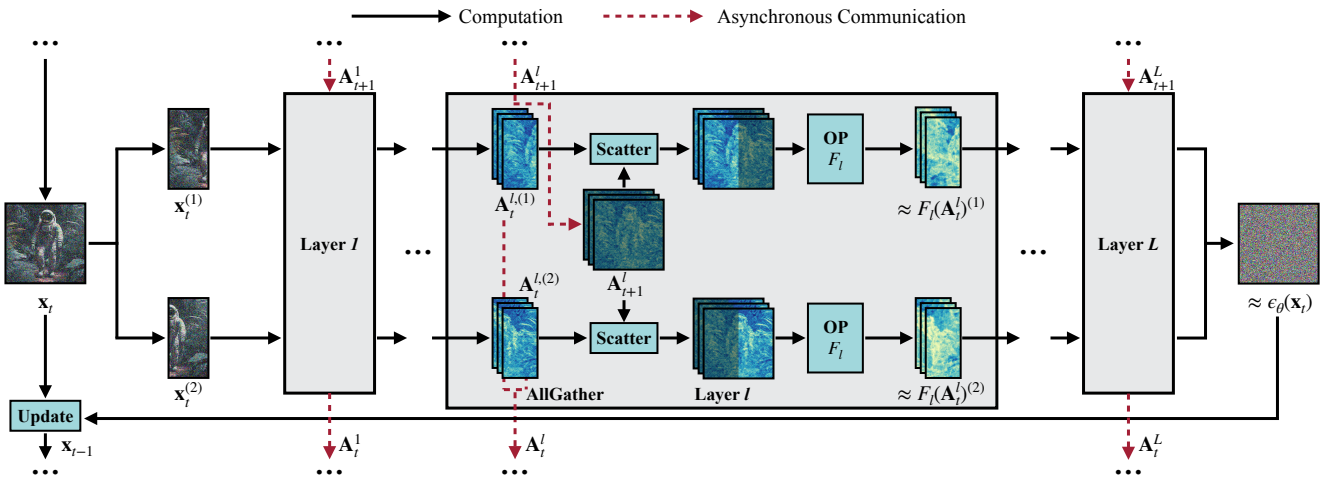

Figure 3. Overview of DistriFusion. For simplicity, we omit the inputs of t and c, and use $N = 2$ devices as an example. Superscripts $^{(1)}$ and $^{(2)}$ represent the first and the second patch, respectively. Stale activations from the previous step are darkened. At each step t, we first split the input $\mathbf{x}_t$ into N patches $\mathbf{x}_t^{(1)}, \dots, \mathbf{x}_t^{(N)}$. For each layer l and device i, upon getting the input activation patches $\mathbf{A}_t^{l,(i)}$, two operations then process asynchronously: First, on device i, $\mathbf{A}_t^{l,(i)}$ is scattered back into the stale activation $\mathbf{A}_{t+1}^l$ from the previous step. The output of this `Scatter` operation is then fed into the sparse operator F_l (linear, convolution, or attention layers), which performs computations exclusively on the fresh regions and produces the corresponding output. Meanwhile, an `AllGather` operation is performed over $\mathbf{A}_t^{l,(i)}$ to prepare the full activation $\mathbf{A}_t^l$ for the next step. We repeat this procedure for each layer. The final outputs are then aggregated together to approximate $\epsilon_\theta(\mathbf{x}_t)$, which is used to compute $\mathbf{x}_{t-1}$. The timeline visualization of each device for predicting $\epsilon_\theta(\mathbf{x}_t)$ is shown in Figure 4.

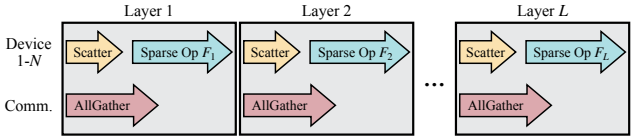

Figure 4. Timeline visualization on each device when predicting $\epsilon_\theta(\mathbf{x}_t)$. *Comm.* means communication, which is asynchronous with computation. The `AllGather` overhead is fully hidden within the computation.

previous step. Then the approximated global mean $\mathbb{E}[\mathbf{A}_t]$ for current step on device i can be computed as

$$\mathbb{E}[\mathbf{A}_t] \approx \underbrace{\mathbb{E}[\mathbf{A}_{t+1}]}_{\text{stale global mean}} + \underbrace{(\mathbb{E}[\mathbf{A}_t^{(i)}] - \mathbb{E}[\mathbf{A}_{t+1}^{(i)}])}_{\text{correction}}. \quad (2)$$

We use the same technique to approximate $\mathbb{E}[(\mathbf{A}_t)^2]$, then the variance can be approximated as $\mathbb{E}[(\mathbf{A}_t)^2] - \mathbb{E}[\mathbf{A}_t]^2$. We then use these approximated statistics for the GN layer and in the meantime aggregate the local mean and variance to compute the precise ones using asynchronous communication. Thus, the communication cost can also be pipelined into the computation. We empirically find this method yields comparable results to the direct synchronous aggregation. However, there are some rare cases where the approximated variance is negative. For these negative variance groups, we will fall back to use the local variance of the fresh patch.

Warm-up steps. As observed in eDiff-I [2] and FastComposer [65], the behavior of diffusion synthesis undergoes qualitative changes throughout the denoising process. Specifically, the initial steps of sampling predominantly shape the low-frequency aspects of the image, such as spatial layout and overall semantics. As the sampling progresses, the focus shifts to recovering local high-frequency details. Therefore, to boost image quality, especially in samplers with a reduced number of steps, we adopt warm-up steps. Instead of directly employing displaced patch parallelism after the first step, we continue with several iterations of the standard synchronous patch parallelism as a preliminary phase, or *warm-up*. As detailed in Section 5.3, this integration of warm-up steps significantly improves performance.

5. Experiments

We first describe our experiment setups, including our benchmark datasets, baselines, and evaluation protocols. Then we present our main results regarding both quality and efficiency. Finally, we further show some ablation studies to verify each design choice.

5.1. Setups

Models. As our method only requires off-the-shelf pre-trained diffusion models, we mainly conduct experiments on the state-of-the-art public text-to-image model Stable Diffusion XL (SDXL) [43]. SDXL first compresses an image to an $8\times$ smaller latent representation using a pre-trained auto-encoder and then applies a diffusion model in this latent space. It also incorporates multiple cross-attention layers to facilitate text conditioning. Compared to the original Stable Diffusion [51], SDXL adopts significantly more attention layers, resulting in a more computationally intensive model.

Datasets. We use the HuggingFace version of COCO Captions 2014 [5] dataset to benchmark our method. This dataset contains human-generated captions for images

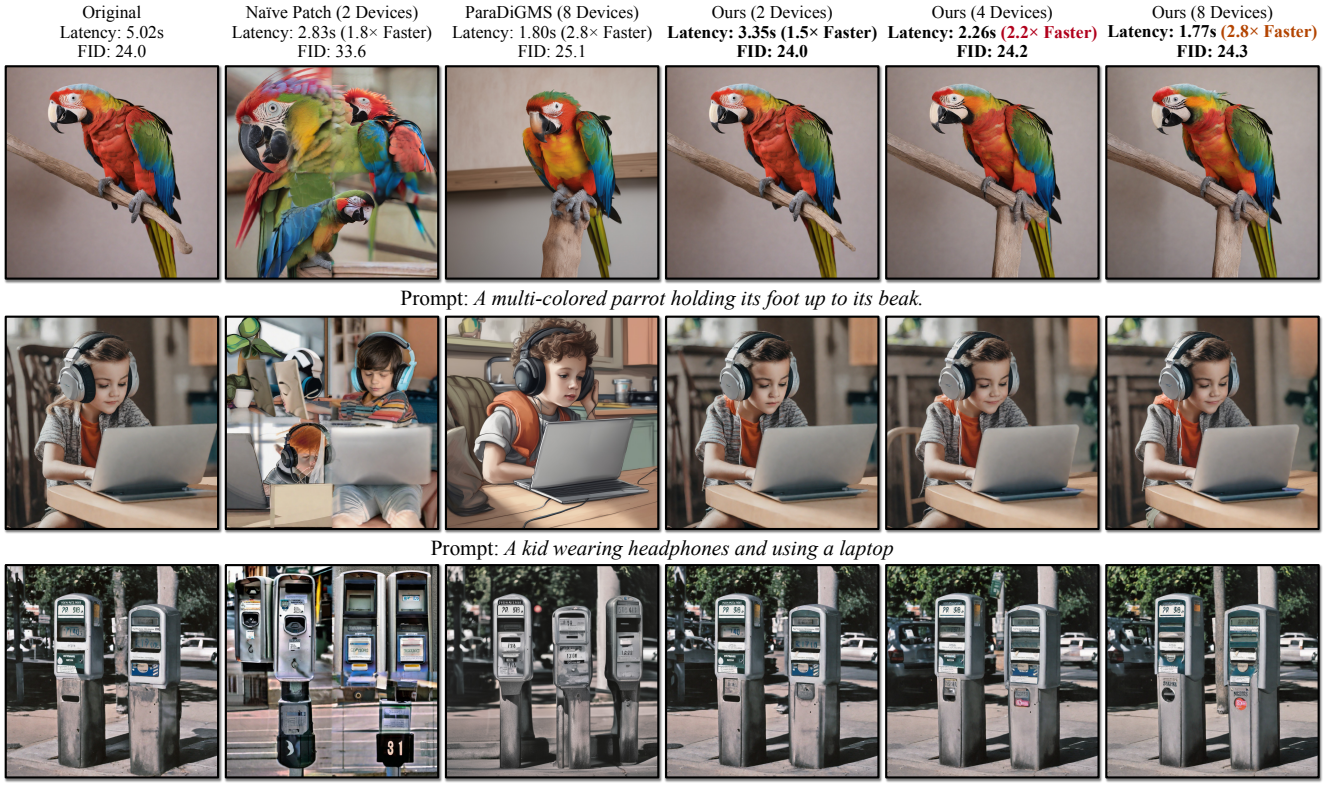

Figure 5. Qualitative results. FID is computed against the ground-truth images. Our DistriFusion can reduce the latency according to the number of used devices while preserving visual fidelity.

from Microsoft Common Objects in COntext (COCO) dataset [30]. For evaluation, we randomly sample a subset from the validation set, which contains 5K images with one caption per image.

Baselines. We compare our DistriFusion against the following baselines in terms of both quality and efficiency:

- *Naïve Patch*. At each iteration, the input is divided row-wise or column-wise alternately. These patches are then processed independently by the model, without any interaction between them. The outputs are subsequently concatenated together.
- *ParaDiGMS [56]* is a technique to accelerate pre-trained diffusion models by denoising multiple steps in parallel. It uses Picard iterations to guess the solution of future steps and iteratively refines it until convergence. We use a batch size 8 for ParaDiGMS to align with Table 4 in the original paper [56]. We empirically find this setting yields the best performance in both quality and latency.

Metrics. Following previous works [22, 23, 35, 41], we evaluate the image quality with standard metrics: Peak Signal Noise Ratio (PSNR, higher is better), LPIPS (lower is better) [72], and Fréchet Inception Distance (FID, lower is better) [11][†]. We employ PSNR to quantify the minor numerical differences between the outputs of the benchmarked method and the original diffusion model outputs. LPIPS is used to evaluate perceptual similarity. Additionally, the FID score is used to measure the distributional differences between the outputs of the method and either the original outputs or the ground-truth images.

Implementation details. By default, we adopt the 50-step DDIM sampler [58] with classifier-free guidance scale 5 to generate 1024×1024 images, unless otherwise specified. In addition to the first step, we perform another 4-step synchronous patch parallelism, serving as a warm-up phase. Please refer to Section 5.1 in our arXiv version for the latency measurement details.

5.2. Main Results

Quality results. In Figure 5, we show some qualitative visual results and report some quantitative evaluation in Table 1. *with G.T.* means computing the metric with the ground-truth COCO [30] images, whereas *w/ Orig.* refers to computing the metrics with the outputs from the original model. For PSNR, we report only the *w/ Orig.* setting, as the *w/ G.T.* comparison is not informative due to significant numerical differences between the generated outputs and the ground-truth images.

As shown in Table 1, ParaDiGMS [56] expends considerable computational resources on guessing future denoising steps, resulting in a much higher total MACs. Besides, it

[†]We use TorchMetrics to calculate PSNR and LPIPS, and use Clean-FID [42] to calculate FID.

#Steps	#Devices	Method	PSNR (↑)	LPIPS (↓)		FID (↓)		MACs (T)	Latency	
				w/ G.T.	w/ Orig.	w/ Orig.	w/ G.T.		Value (s)	Speedup
50	1	Original	–	0.797	–	24.0	–	338	5.02	–
	2	Naïve Patch	28.2	0.812	0.596	33.6	29.4	**322**	**2.83**	**1.8**×
		Ours	**31.9**	**0.797**	**0.146**	**24.2**	**4.86**	338	3.35	1.5×
	4	Naïve Patch	27.9	0.853	0.753	125	133	**318**	**1.74**	**2.9**×
		Ours	**31.0**	**0.798**	**0.183**	**24.2**	**5.76**	338	2.26	2.2×
	8	Naïve Patch	27.8	0.892	0.857	252	259	**324**	**1.27**	**4.0**×
		ParaDiGMS	29.3	0.800	0.320	25.1	10.8	657	1.80	2.8×
		Ours	**30.5**	**0.799**	**0.211**	**24.4**	**6.46**	338	1.77	2.8×

Table 1. Quantitative evaluation. *MACs* measures cumulative computation across all devices for the whole denoising process for generating a single 1024 × 1024 image. *w/ G.T.* means calculating the metrics with the ground-truth images, while *w/ Orig.* means with the original model's samples. For PSNR, we report the *w/ Orig.* setting. Our method mirrors the results of the original model across all metrics while maintaining the total MACs. It also reduces the latency on NVIDIA A100 GPUs in proportion to the number of used devices.

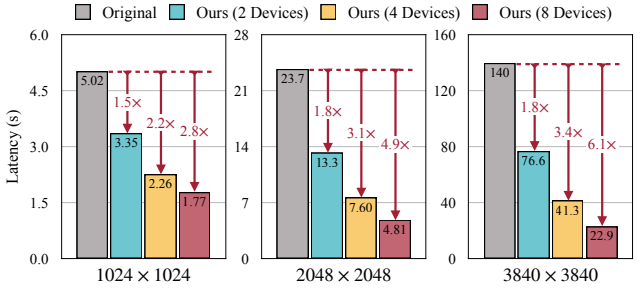

Figure 6. Measured total latency of DistriFusion with the 50-step DDIM sampler [58] for generating a single image across different resolutions on NVIDIA A100 GPUs. When scaling up the resolution, the GPU devices are better utilized. Remarkably, when generating 3840 × 3840 images, DistriFusion achieves 1.8×, 3.4× and 6.1× speedups with 2, 4, and 8 A100s, respectively.

also suffers from some performance degradation. In contrast, our method simply distributes workloads across multiple GPUs, maintaining a constant total computation. The Naïve Patch baseline, while lower in total MACs, lacks the crucial inter-patch interaction, leading to fragmented outputs. This limitation significantly impacts image quality, as reflected across all evaluation metrics. Our DistriFusion can well preserve interaction. Even when using 8 devices, it achieves comparable PSNR, LPIPS, and FID scores comparable to those of the original model.

Speedups. Compared to the theoretical computation reduction, on-hardware acceleration is more critical for real-world applications. To demonstrate the effectiveness of our method, we also report the end-to-end latency in Table 1 on 8 NVIDIA A100 GPUs. In the 50-step setting, ParaDiGMS achieves an identical speedup of 2.8× to our method at the cost of compromised image quality (see Figure 5). In our arXiv version, we also show the speedups on more commonly used 25-step setting. ParaDiGMS only has a marginal 1.3× speedup due to excessive wasted guesses, which is also reported in Shih *et al.* [56]. However, our method can still mirror the original quality and accelerate the model by 2.7×.

Method	1024 × 1024		2048 × 2048		3840 × 3840	
	Comm.	Latency	Comm.	Latency	Comm.	Latency
Original	–	5.02s	–	23.7s	–	140s
Sync. TP	1.33G	3.61s	5.33G	11.7s	18.7G	46.3s
Sync. PP	0.42G	2.21s	1.48G	5.62s	5.38G	24.7s
DistriFusion (Ours)	**0.42G**	**1.77s**	**1.48G**	**4.81s**	**5.38G**	**22.9s**
No Comm.	–	1.48s	–	4.14s	–	21.3s

Table 2. Communication cost comparisons with 8 A100s across different resolutions. *Sync. TP/PP*: Synchronous tensor/patch parallelism. *No Comm.*: An ideal no communication PP. *Comm.* measures the total communication amount. PP only requires less than $\frac{1}{3}$ communication amounts compared to TP. Our DistriFusion further reduces the communication overhead by $50 \sim 60\%$.

When generating 1024 × 1024 images, our speedups are limited by the low GPU utilization of SDXL. To maximize device usage, we further scale the resolution to 2048 × 2048 and 3840 × 3840 in Figure 6. At these larger resolutions, the GPU devices are better utilized. Specifically, for 3840 × 3840 images, DistriFusion reduces the latency by 1.8×, 3.4× and 6.1× with 2, 4 and 8 A100s, respectively. Note that these results are benchmarked with PyTorch. With more advanced compilers, such as TVM [4] and TensorRT [1], we anticipate even higher GPU utilization and consequently more pronounced speedups from DistriFusion, as observed in SIGE [23]. In practical use, the batch size often doubles due to classifier-free guidance [12]. We can first split the batch and then apply DistriFusion to each batch separately. This approach further improves the total speedups to 3.6× and 6.6× with 4 and 8 A100s for generating a single 3840 × 3840 image, respectively.

5.3. Ablation Study

Compare to tensor parallelism. In Table 2, we benchmark our latency with synchronous tensor parallelism (*Sync. TP*) and synchronous patch parallelism (*Sync. PP*), and report the corresponding communication amounts. Compared to TP, PP has better independence, which eliminates the need for communication within cross-attention and linear

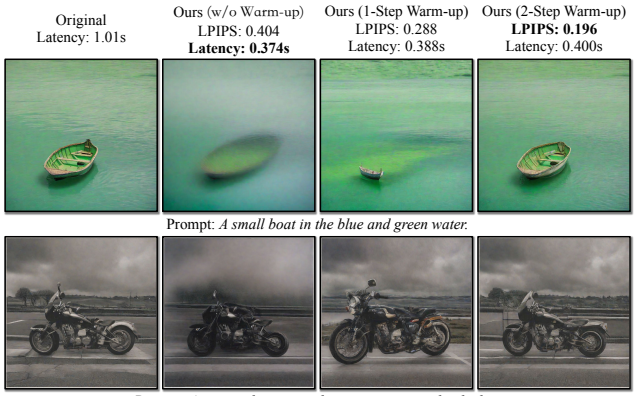

Figure 7. Qualitative results on the 10-step DPM-Solver [32, 33] with different warm-up steps. LPIPS is computed against the samples from the original SDXL over the entire COCO [5] dataset. Naïve DistriFusion without warm-up steps has evident quality degradation. Adding a 2-step warm-up significantly improves the performance while avoiding high latency rise.

layers. For convolutional layers, communication is only required at the patch boundaries, which represent a minimal portion of the entire tensor. Moreover, PP utilizes `AllGather` over `AllReduce`, leading to lower communication demands and no additional use of computing resources. Therefore, PP requires 60% fewer communication amounts and is $1.6 \sim 2.1\times$ faster than TP, making it a more efficient approach for deploying diffusion models. We also include a theoretical PP baseline without any communication (*No Comm.*) to demonstrate the communication overhead in *Sync. PP* and DistriFusion. Compared to *Sync. PP*, DistriFusion further cuts such overhead by over 50%. The remaining overhead mainly comes from our current usage of NVIDIA Collective Communication Library (NCCL) for asynchronous communication. NCCL kernels use SMs (the computing resources on GPUs), which will slow down the overlapped computation. Using remote memory access can bypass this issue and close the performance gap.

Few-step sampling and warm-up steps. Our approach hinges on the observation that adjacent denoising steps share similar inputs, *i.e.*, $\mathbf{x}_t \approx \mathbf{x}_{t-1}$. However, as we increase the step size and thereby reduce the number of steps, the approximation error escalates, potentially compromising the effectiveness of our method. In Figure 7, we present results using 10-step DPM-Solver [32, 33]. The 10-step configuration is the threshold for the training-free samplers to maintain the image quality. Under this setting, naïve DistriFusion without warm-up struggles to preserve the image quality. However, incorporating an additional two-step warm-up significantly recovers the performance with only slightly increased latency.

GroupNorm. As discussed in Section 4, calculating accurate group normalization (GN) statistics is crucial for preserving image quality. In Figure 8, we compare four different

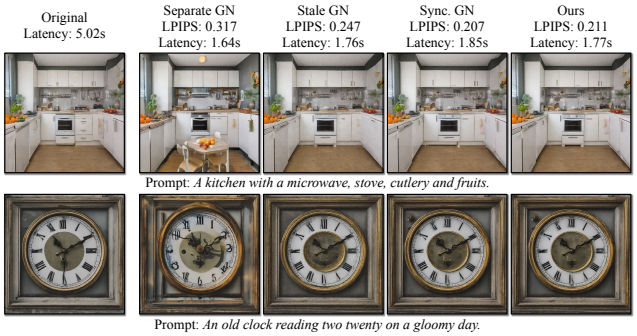

Figure 8. Qualitative results of different GN schemes with 8 A100s. LPIPS is computed against the original samples over the whole COCO [5] dataset. *Separate GN* only utilizes the statistics from the on-device patch. *Stale GN* reuses the stale statistics. They suffer from quality degradation. *Sync. GN* synchronizes data to ensure accurate statistics at the cost of extra overhead. Our corrected asynchronous GN, by correcting stale statistics, avoids the need for synchronization and effectively restores quality.

GN schemes. The first approach *Separate GN* uses statistics from the on-device fresh patch. This approach delivers the best speed at the cost of lower image fidelity. This compromise is particularly severe for large numbers of used devices, due to insufficient patch size for precise statistics estimation. The second scheme *Stale GN* computes statistics using stale activations. However, this method also faces quality degradation, because of the different distributions between stale and fresh activations, often resulting in images with a fog-like noise effect. The third approach *Sync. GN* use synchronized communication to aggregate accurate statistics. Though achieving the best image quality, it suffers from large synchronization overhead. Our method uses a correction term to close the distribution gap between the stale and fresh statistics. It achieves image quality on par with *Sync. GN* but without incurring synchronous communication overhead.

6. Conclusion & Discussion

In this paper, we introduce DistriFusion to accelerate diffusion models with multiple GPUs for parallelism. Our method divides images into patches, assigning each to a separate GPU. We reuse the pre-computed activations from previous steps to maintain patch interactions. On Stable Diffusion XL, our method achieves up to a $6.1\times$ speedup on 8 NVIDIA A100s. This advancement not only enhances the efficiency of AI-generated content creation but also sets a new benchmark for future research in parallel computing for AI applications.

Acknowledgments

We thank Jun-Yan Zhu and Ligeng Zhu for their helpful discussion and valuable feedback. The project is supported by MIT-IBM Watson AI Lab, Amazon, MIT Science Hub, and National Science Foundation.

References

[1] *NVIDIA/TensorRT*. 2023. 7

[2] Yogesh Balaji, Seungjun Nah, Xun Huang, Arash Vahdat, Jiaming Song, Karsten Kreis, Miika Aittala, Timo Aila, Samuli Laine, Bryan Catanzaro, et al. ediffi: Text-to-image diffusion models with an ensemble of expert denoisers. *arXiv preprint arXiv:2211.01324*, 2022. 1, 2, 5

[3] James Betker, Gabriel Goh, Li Jing, Tim Brooks, Jianfeng Wang, Linjie Li, Long Ouyang, Juntang Zhuang, Joyce Lee, Yufei Guo, et al. Improving image generation with better captions. *Computer Science. https://cdn.openai.com/papers/dall-e-3.pdf*, 2023. 1

[4] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In *OSDI*, 2018. 7

[5] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*, 2015. 2, 5, 8

[6] Xiaoliang Dai, Ji Hou, Chih-Yao Ma, Sam Tsai, Jialiang Wang, Rui Wang, Peizhao Zhang, Simon Vandenhende, Xiaofang Wang, Abhimanyu Dubey, et al. Emu: Enhancing image generation models using photogenic needles in a haystack. *arXiv preprint arXiv:2309.15807*, 2023. 1

[7] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *CVPR*, 2017. 3

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *NeurIPS*, 2014. 3

[9] Jiatao Gu, Shuangfei Zhai, Yizhe Zhang, Josh Susskind, and Navdeep Jaitly. Matryoshka diffusion models. *arXiv preprint arXiv:2310.15111*, 2023. 3

[10] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *NeurIPS*, 2015. 3

[11] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *NeurIPS*, 2017. 6

[12] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021. 2, 7

[13] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *NeurIPS*, 2020. 1, 2

[14] Emiel Hoogeboom, Jonathan Heek, and Tim Salimans. simple diffusion: End-to-end diffusion for high resolution images. *arXiv preprint arXiv:2301.11093*, 2023. 3

[15] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *NeurIPS*, 2019. 3

[16] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014. 3

[17] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. *MLSys*, 2019. 3

[18] Patrick Judd, Alberto Delmas, Sayeh Sharify, and Andreas Moshovos. Cnvlutin2: Ineffectual-activation-and-weight-free deep neural network computing. *arXiv preprint arXiv:1705.00125*, 2017. 3

[19] Gwanghyun Kim and Jong Chul Ye. Diffusionclip: Text-guided image manipulation using diffusion models. *arXiv preprint arXiv:2110.02711*, 2021. 3

[20] Zhifeng Kong and Wei Ping. On fast sampling of diffusion probabilistic models. In *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*, 2021. 2

[21] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *ICLR*, 2016. 3

[22] Muyang Li, Ji Lin, Yaoyao Ding, Zhijian Liu, Jun-Yan Zhu, and Song Han. Gan compression: Efficient architectures for interactive conditional gans. In *CVPR*, 2020. 6

[23] Muyang Li, Ji Lin, Chenlin Meng, Stefano Ermon, Song Han, and Jun-Yan Zhu. Efficient spatially sparse inference for conditional gans and diffusion models. In *NeurIPS*, 2022. 2, 3, 4, 6, 7

[24] Xiaoxiao Li, Ziwei Liu, Ping Luo, Chen Change Loy, and Xiaoou Tang. Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade. In *CVPR*, 2017. 3

[25] Xiuyu Li, Long Lian, Yijiang Liu, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-diffusion: Quantizing diffusion models. *arXiv preprint arXiv:2302.04304*, 2023. 2, 3

[26] Yanyu Li, Huan Wang, Qing Jin, Ju Hu, Pavlo Chemerys, Yun Fu, Yanzhi Wang, Sergey Tulyakov, and Jian Ren. Snapfusion: Text-to-image diffusion model on mobile devices within two seconds. *NeurIPS*, 2023. 2, 3

[27] Zhuohan Li, Siyuan Zhuang, Shiyuan Guo, Danyang Zhuo, Hao Zhang, D. Song, and I. Stoica. Terapipe: Token-level pipeline parallelism for training large-scale language models. *ICML*, 2021. 3

[28] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Z. Chen, Hao Zhang, Joseph E. Gonzalez, and I. Stoica. Alpaserve: Statistical multiplexing with model parallelism for deep learning serving. *USENIX Symposium on Operating Systems Design and Implementation*, 2023. 3

[29] Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, and Song Han. Mcunetv2: Memory-efficient patch-based inference for tiny deep learning. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021. 3

[30] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014. 6

[31] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *CVPR*, 2015. 3

[32] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *arXiv preprint arXiv:2206.00927*, 2022. 2, 3, 8

[33] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *arXiv preprint arXiv:2211.01095*, 2022. 8

[34] Chenlin Meng, Ruiqi Gao, Diederik P Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. *arXiv preprint arXiv:2210.03142*, 2022. 2, 3

[35] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. SDEdit: Guided image synthesis and editing with stochastic differential equations. In *ICLR*, 2022. 6

[36] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *SOSP*, 2019. 3

[37] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, J. Bernauer, Bryan Catanzaro, Amar Phanishayee, and M. Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm. *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021. 3

[38] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *ICML*, 2021. 4

[39] Alexander Quinn Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob Mcgrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. In *ICML*, 2022. 2

[40] Bowen Pan, Wuwei Lin, Xiaolin Fang, Chaoqin Huang, Bolei Zhou, and Cewu Lu. Recurrent residual module for fast inference in videos. In *CVPR*, 2018. 3

[41] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *CVPR*, 2019. 6

[42] Gaurav Parmar, Richard Zhang, and Jun-Yan Zhu. On aliased resizing and surprising subtleties in GAN evaluation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 11400–11410. IEEE, 2022. 6

[43] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. In *ICLR*, 2024. 1, 2, 3, 5

[44] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. *Sc20: International Conference For High Performance Computing, Networking, Storage And Analysis*, 2019. 3

[45] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *ICML*, 2021. 1

[46] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022. 1

[47] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506, 2020. 3

[48] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. Zero-offload: Democratizing billion-scale model training. In *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*, pages 551–564. USENIX Association, 2021. 3

[49] Mengye Ren, Andrei Pokrovsky, Bin Yang, and Raquel Urtasun. Sbnet: Sparse blocks network for fast inference. In *CVPR*, 2018. 3

[50] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *CVPR*, 2017. 3

[51] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022. 1, 3, 5

[52] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015. 3

[53] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *NeurIPS*, 2022. 1

[54] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *ICLR*, 2021. 2, 3

[55] Shaohuai Shi and Xiaowen Chu. Speeding up convolutional neural networks by exploiting the sparsity of rectifier units. *arXiv preprint arXiv:1704.07724*, 2017. 3

[56] Andy Shih, Suneel Belkhale, Stefano Ermon, Dorsa Sadigh, and Nima Anari. Parallel sampling of diffusion models. *NeurIPS*, 2023. 3, 4, 6, 7

[57] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, 2015. 1

[58] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*, 2020. 2, 6, 7

[59] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *ICLR*, 2020. 3

[60] Haotian Tang, Zhijian Liu, Xiuyu Li, Yujun Lin, and Song Han. Torchsparse: Efficient point cloud inference engine. In *MLSys*, 2022. 3

[61] Haotian Tang, Shang Yang, Zhijian Liu, Ke Hong, Zhongming Yu, Xiuyu Li, Guohao Dai, Yu Wang, and Song Han. Torchsparse++: Efficient training and inference framework for sparse convolution on gpus. In *MICRO*, 2023. 3

[62] Arash Vahdat, Karsten Kreis, and Jan Kautz. Score-based generative modeling in latent space. 34:11287–11302, 2021. 3

[63] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990. 3

[64] Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018. 4

[65] Guangxuan Xiao, Tianwei Yin, William T. Freeman, Frédo Durand, and Song Han. Fastcomposer: Tuning-free multi-subject image generation with localized attention. *arXiv*, 2023. 5

[66] Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. Tackling the generative learning trilemma with denoising diffusion GANs. In *ICLR*, 2022. 2

[67] Yuanzhong Xu, HyoukJoong Lee, Dehao Chen, Blake Hechtman, Yanping Huang, Rahul Joshi, Maxim Krikun, Dmitry Lepikhin, Andy Ly, Marcello Maggioni, Ruoming Pang, Noam Shazeer, Shibo Wang, Tao Wang, Yonghui Wu, and Zhifeng Chen. Gspmd: General and scalable parallelization for ml computation graphs. *arXiv preprint arXiv: 2105.04663*, 2021. 3

[68] Jinhui Yuan, Xinqi Li, Cheng Cheng, Juncheng Liu, Ran Guo, Shenghang Cai, Chi Yao, Fei Yang, Xiaodong Yi, Chuan Wu, Haoran Zhang, and Jie Zhao. Oneflow: Redesign the distributed deep learning framework from scratch. *arXiv preprint arXiv: 2110.15032*, 2021. 3

[69] Qinsheng Zhang and Yongxin Chen. Fast sampling of diffusion models with exponential integrator. In *ICLR*, 2022. 2, 3

[70] Qinsheng Zhang, Molei Tao, and Yongxin Chen. gddim: Generalized denoising diffusion implicit models. 2022. 3

[71] Qinsheng Zhang, Jiaming Song, Xun Huang, Yongxin Chen, and Ming yu Liu. Diffcollage: Parallel generation of large content with diffusion models. In *CVPR*, 2023. 3

[72] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 6

[73] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023. 3

[74] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P Xing, et al. Alpa: Automating inter- and {Intra-Operator} parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 559–578, 2022. 3