DICE: Staleness-Centric Optimizations for Parallel Diffusion MoE Inference

Supplementary Material

7. More Preliminaries

7.1. Mixture-of-Experts.

Figure 11 shows the architecture of MoE model. The router directs each token to a subset of experts based on router scores, while a shared expert captures common representations. Outputs from the activated experts are then combined to form the final output. This design allows sublinear scaling by selectively activating only a few experts per input.

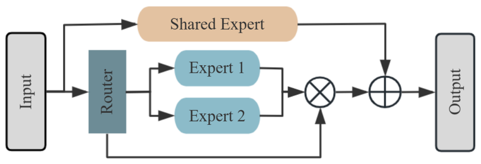

Figure 11. Illustration of an MoE model[7] with one shared experts and two non-shared experts.

7.2. Displaced Parallelism.

Displaced parallelism [22] overlaps communication and computation by asynchronously transmitting activations computed in the current step for use in the next step (Algorithm 2). this prevents blocking (Algorithm 1). As shown in Figure 12, each device sends activations without waiting, continuing computation with slightly outdated activations from the previous step. This approach prevents communication-induced blocking but introduces staleness, as computations rely on slightly outdated activations rather than fresh data.

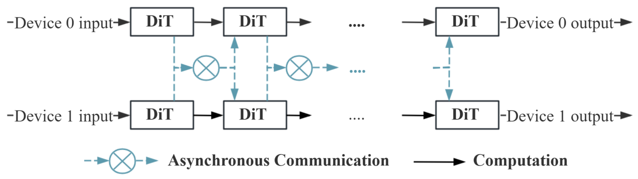

Figure 12. Illustration of displaced parallelism in DiT across multiple devices, with dashed arrows representing asynchronous communication steps that defer data exchange until the next computation stage.

Algorithm 1: Expert Parallelism

```
1 for each layer do
      // Perform attention
2     $x \leftarrow \text{attn}(x)$
3     $[s, idx] \leftarrow \text{router}(x)$ // Compute router scores and assign tokens to experts
4     $x_d \leftarrow \text{all2all\_dispatch}(x, idx)$ // Synchronous dispatch (blocking)
5     $x_e \leftarrow \text{expert}(x_d)$ // Process tokens with local experts
6     $x_c \leftarrow \text{all2all\_combine}(x_e)$ // Synchronous combine (blocking)
7     $x \leftarrow \text{scale}(x_c, s)$ // Scale outputs using router scores
```

Algorithm 2: Displaced Expert Parallelism

```
1 for each layer do
2     $x \leftarrow \text{attn}(x)$
3     $[s, idx] \leftarrow \text{router}(x)$
4     $h_d \leftarrow \text{async\_dispatch}(x, idx)$ // Launch non-blocking dispatch (returns handle $h_d$)
5     $x_d \leftarrow \text{wait}(h_d^{prev})$ // Wait for previous step's dispatch result
6     $x_e \leftarrow \text{expert}(x_d)$ // Process outdated tokens with local experts
7     $h_c \leftarrow \text{async\_combine}(x_e)$ // Launch non-blocking combine (returns handle $h_c$)
8     $x_c \leftarrow \text{wait}(h_c^{prev})$ // Wait for previous step's combine result
9     $x \leftarrow \text{scale}(x_c, s)$
```

8. Alternative Staggered Batch Solution

Many works split batches to better overlap communication and computation [11, 20]. We initially explored a **staggered batch solution**, in which each device divides its local batch into multiple sub-batches and processes them in a staggered manner. As shown in Figure 13, this approach can reduce staleness by handling multiple sub-batches.

However, we ultimately do not adopt this solution for three main reasons:

- **Reduced GPU Utilization.** Splitting the batch lowers the effective batch size per device, potentially underutilizing GPU resources and degrading throughput.

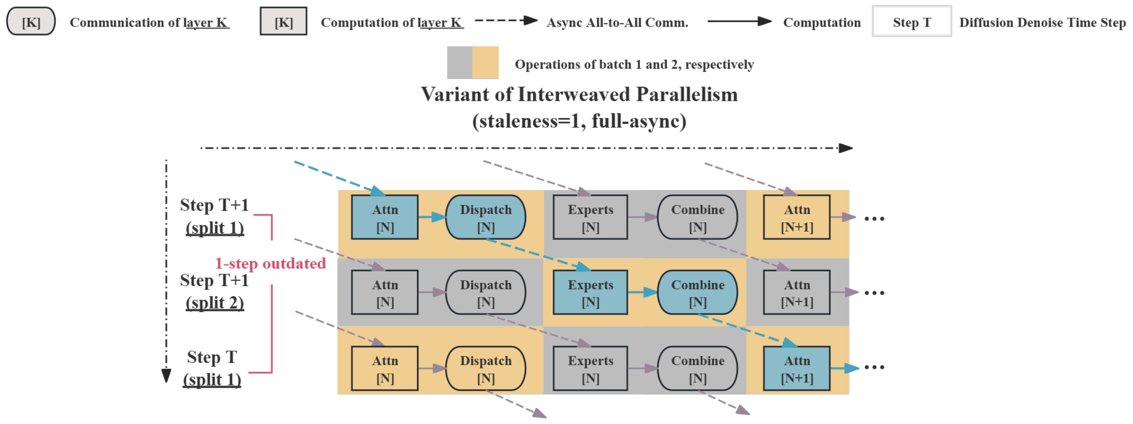
[K] Communication of layer K
[K] Computation of layer K
Async All-to-All Comm.
Computation
Step T Diffusion Denoise Time Step
Operations of batch 1 and 2, respectively
Variant of Interweaved Parallelism (staleness=1, full-async)
Step T+1 (split 1)
1-step outdated
Step T+1 (split 2)
Step T (split 1)
Attn [N]
Dispatch [N]
Experts [N]
Combine [N]
Attn [N+1]

Figure 13. Illustration to the *staggered batch solution*, where local batches are split and interleaved to reduce staleness. However, this approach requires additional buffers, doubles memory usage, and reduces GPU utilization due to smaller effective batch sizes.

- **Increased Memory Requirement.** Unlike interweaved parallelism—which requires a persistent buffer only for combine and a temporary one for dispatch—the staggered batch solution must maintain persistent buffers for both dispatch and combine, effectively doubling memory usage and increasing the risk of out-of-memory.
- **Batch Size Limitations.** This solution requires a local batch size greater than one, which is not feasible when processing a single large image or in scenarios with limited batch capacity.

For these reasons, we prioritize the interweaved parallelism approach described earlier, which retains the benefits of asynchronous scheduling while requiring fewer buffers and achieving higher GPU utilization.

Model	GPUs	Batch Size			
		4	8	16	32
DiT-MoE-XL	4	62.9%	70.9%	73.8%	74.4%
	8	75.6%	78.1%	79.0%	79.2%
DiT-MoE-G	4	50.7%	56.8%	59.3%	61.1%
	8	64.7%	67.8%	69.2%	68.9%

Table 5. All-to-All communication time percentage in synchronous expert parallelism.

9. More Implementation Details

Setup. The models used in our study are the publicly available versions of DiT-MoE from Huggingface. In our experimental results, the batch size refers to the local batch size, representing the number of samples processed per device. All asynchronous methods apply the same number of synchronized steps(e.g. warmup) post-start.

Expert Score Scaling. There are two approaches for scaling results after expert processing: (1) using the latest router scores computed in the current step, which provides fresher scores, and (2) using the router scores corresponding to the stale expert input, offering better alignment with the activations used. The selection of the scores has little impact on performance. For fairness, both displaced parallelism and DICE use the stale router scores for scaling.

Extending Image Sizes. The public DiT-MoE model supports relatively low resolution image dimensions. To extend our experiments to larger images, we initialize positional embeddings for other sizes. Although this adjustment prevents the model from generating meaningful images, it enables us to evaluate latency, memory usage, and speedup across different resolutions.

Interweaved Parallelism restructures the execution flow, effectively “folding” it. It processes expert computations within the same step while maintaining asynchronous communication, as outlined in Algorithm 3.

Conditional Communication For each token-expert pair (i, e) at step t, conditional communication dynamically decides whether to send fresh activations or reuse cached results. As shown in Algorithm 4, the top-1 expert for each token always receives fresh data, ensuring critical tokens remain up-to-date. Lower-priority experts (i.e., those not ranked top-1) reuse their previous activations most of the time and only receive an update every n steps. This design leverages the weighted-sum mechanic of MoE to maintain high-impact tokens’ freshness while reducing communication overhead for less critical tokens.

10. Discussion

Limitations. Although DICE demonstrates significant gains in inference efficiency, there remain avenues for further improvements. Optimized kernels and more efficient NCCL operations could help further reduce latency. Addi-

Algorithm 3: Interweaved Parallelism

```
1 for each layer do
2     $x \leftarrow$ attn($x$)
3     $[s, idx] \leftarrow$ router($x$)
4     $h_d \leftarrow$ async_dispatch($x, idx$) // Launch
          non-blocking dispatch for
          current layer
5     prev_layer_x$_d \leftarrow$ wait(prev_layer_h$_d$) // Wait
          for previous layer's dispatch
          result from current step
6     prev_layer_x$_e \leftarrow$
          prev_layer_expert(prev_layer_x$_d$)
          // Process tokens using
          previous layer's experts
7     prev_layer_h$_c \leftarrow$ async_combine(prev_layer_x$_e$)
          // Launch non-blocking
          combine for previous layer's
          outputs
8     x$_c \leftarrow$ wait($h_c^{\text{prev}}$) // Wait for previous
          step's combine result for
          current layer
9     $x \leftarrow$ scale(x$_c, s$) // Scale outputs
          using router scores
```

Algorithm 4: Conditional Communication

```
1 for token-expert pair $(i, e)$ at step $t$ do
2     if $e$ is top-1 expert then
3         Transmit Token ;     // Always fresh
4     else
5         if $t \bmod n \neq 0$ then
6             Reuse stale $\mathbf{h}_i^e$;
7         else
8             Transmit Token;
              ;    // Update every $n$ steps
```

tionally, integrating DICE with existing expert parallelism optimizations offers opportunities to enhance its efficiency and scalability. Another limitation lies in the availability of MoE-based diffusion models, which restricts our evaluations to a limited set of configurations. As more MoE-based diffusion models are developed, DICE can be validated and refined across a broader range of scenarios.

Influence of Shared Experts. The architecture of DiT-MoE includes shared experts, a proven mechanism for enhancing MoE performance. We hypothesize that these shared experts may help mitigate the impact of staleness in similarity-based asynchronous parallelism. Unlike routed experts, whose outputs can become stale, the shared expert's computations are always up-to-date (as they are duplicated across devices), potentially providing fresh information to balance the delayed outputs from routed experts. This characteristic might play a role in the performance of DICE, particularly when compared to DistriFusion. While both approaches exhibit a staleness of 1, DICE confines staleness to routed experts while benefiting from the shared expert's fresh contributions. This suggests a possible advantage for DICE in MoE-based models.

Applicability to NVLink. While our experiments are conducted on PCIe-connected GPUs, DICE remains applicable when MoE models are served under NVLink and InfiniBand-based multi-node deployments [44]. In such settings, all-to-all communication can contribute up to **76%** of total inference latency [44], suggesting that DICE might offer even greater benefits in these environments.

Integration with Existing Expert Parallelism Optimizations. Existing expert parallelism optimizations—such as expert shadowing [11], topology-aware routing [11, 21], and affinity-based methods [44]—address orthogonal challenges; respectively: load balancing, network topology, and expert placement. These techniques are **potentially integrable with DICE** rather than alternatives, and combining them could further enhance overall efficiency.

Usage of Router Similarity. Routers assign token destinations during all-to-all communication, and their inherent redundancy is crucial to maintain consistent token-to-expert assignments. Without this similarity, asynchronous (displaced) operations would disrupt token-expert assignments and degrade performance.

11. Additional Experiments

All-to-All Blocking Latency in Expert Parallelism. We measure all-to-all latency on the same hardware as Section 5. Table 5 reveals a significant bottleneck inherent in synchronous Expert Parallelism for DiT-MoE models: the all-to-all communication overhead. The experimental data indicates that, across all tested configurations, all-to-all communication time significantly surpasses half of the total inference time. Furthermore, a pronounced upward trend is observed in this percentage as the batch size increases, reaching a strikingly high 79.2%. This underscores the necessity of our proposed approach to mitigate this communication bottleneck and achieve substantial inference acceleration.

Quality results. In Figure 14, we present additional qualitative results across six different classes. Notably, only the displaced expert parallelism without extra synchronization demonstrates considerable deviations. In particular, the images produced by DICE closely resemble the synced version.

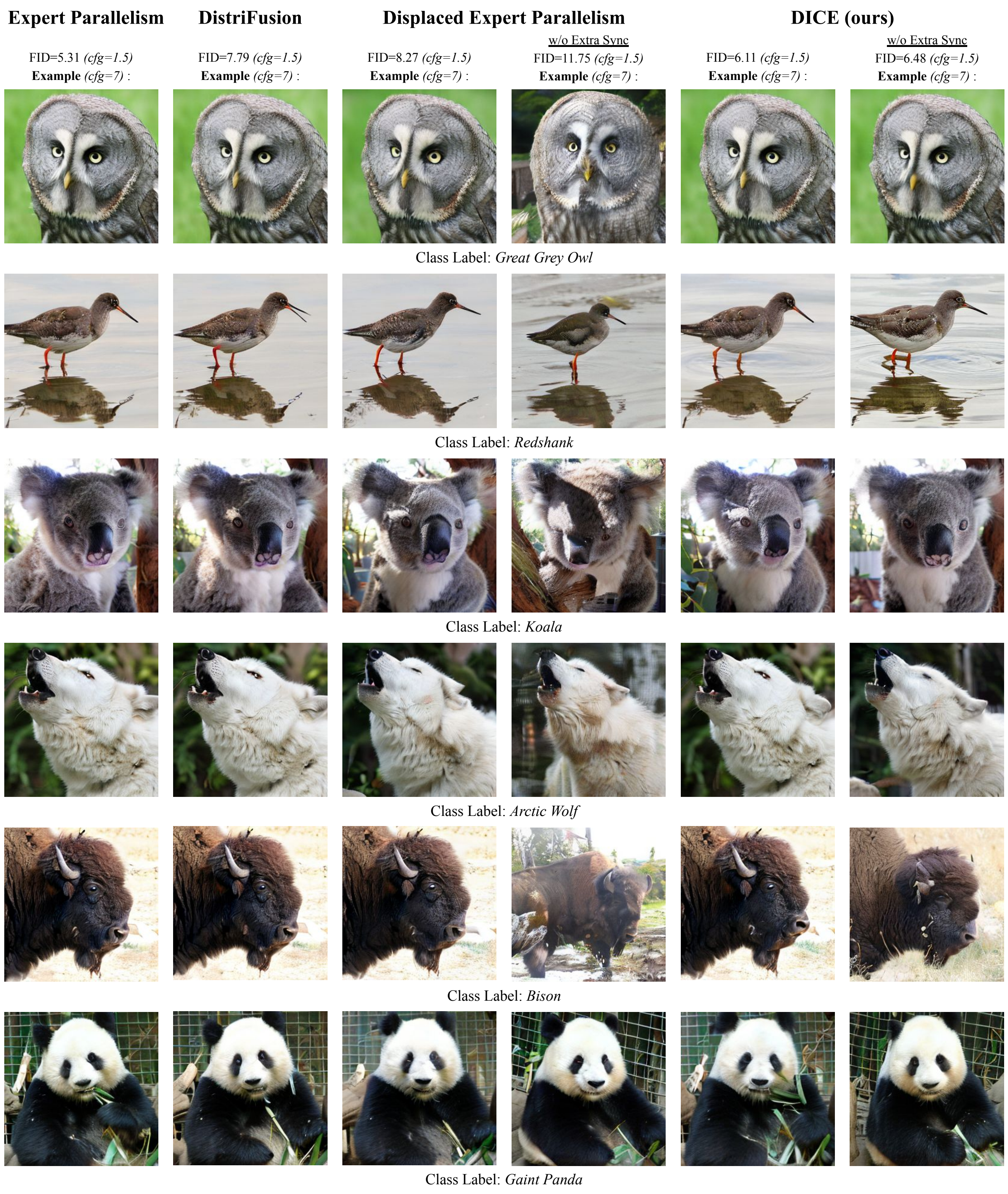

Figure 14. Extra qualitative results.