

---

# TUTEL: ADAPTIVE MIXTURE-OF-EXPERTS AT SCALE

---

Changho Hwang<sup>\*1</sup> Wei Cui<sup>\*1</sup> Yifan Xiong<sup>\*1</sup> Ziyue Yang<sup>\*1</sup> Ze Liu<sup>1</sup> Han Hu<sup>1</sup> Zilong Wang<sup>2</sup> Rafael Salas<sup>2</sup>  
Jithin Jose<sup>2</sup> Prabhat Ram<sup>2</sup> Joe Chau<sup>2</sup> Peng Cheng<sup>1</sup> Fan Yang<sup>1</sup> Mao Yang<sup>1</sup> Yongqiang Xiong<sup>1</sup>

## ABSTRACT

Sparsely-gated mixture-of-experts (MoE) has been widely adopted to scale deep learning models to trillion-plus parameters with fixed computational cost. The algorithmic performance of MoE relies on its token routing mechanism that forwards each input token to the right sub-models or *experts*. While token routing dynamically determines the amount of expert workload at runtime, existing systems suffer inefficient computation due to their *static execution*, namely static parallelism and pipelining, which does not adapt to the dynamic workload.

We present TUTEL, a highly scalable stack design and implementation for MoE with dynamically adaptive parallelism and pipelining. TUTEL designs an identical layout for distributing MoE model parameters and input data, which can be leveraged by switchable parallelism and dynamic pipelining methods without mathematical inequivalence or tensor migration overhead. This enables adaptive parallelism/pipelining optimization at *zero cost* during runtime. Based on this key design, TUTEL also implements various MoE acceleration techniques including Flexible All-to-All, two-dimensional hierarchical (2DH) All-to-All, fast encode/decode, etc. Aggregating all techniques, TUTEL finally delivers **4.96×** and **5.75×** speedup of a single MoE layer over 16 and 2,048 A100 GPUs, respectively, over the previous state-of-the-art.

Our evaluation shows that TUTEL efficiently and effectively runs a real-world MoE-based model named SwinV2-MoE, built upon Swin Transformer V2, a state-of-the-art computer vision architecture. On efficiency, TUTEL accelerates SwinV2-MoE, achieving up to 1.55× and 2.11× speedup in training and inference over Fairseq, respectively. On effectiveness, the SwinV2-MoE model achieves superior accuracy in both pre-training and down-stream computer vision tasks such as COCO object detection than the counterpart dense model, indicating the readiness of TUTEL for end-to-end real-world model training and inference.

## 1 INTRODUCTION

In recent years, the community has found that enrolling more model parameters is one of the most straight-forward but less sophisticated way to improve the performance of deep learning (DL) algorithms (Kaplan et al., 2020). However, model capacity is often limited by computing resource and energy cost (Sharir et al., 2020). To tackle this, sparsely-gated Mixture-of-Experts (MoE) (Shazeer et al., 2017) introduces a *sparse* architecture by employing multiple parallel sub-models called *experts*, where each input is only forwarded to a few experts based on an intelligent gating function. Unlike dense layers, this method scales the model capacity up at only sublinearly increasing computational cost. Nowadays, MoE is one of the most popular approaches demonstrated to scale DNNs to trillion-plus parameters (Fedus et al., 2022), paving the way for models capable of learning even more information.

While MoE-based algorithms open up a huge scale-up/out opportunity, the **dynamic nature of MoE** introduces fundamental system-side challenges that have not been seen before in most of previous DL algorithms and systems. To be specific, each MoE layer consists of a certain number of parallel experts that are distributed over accelerators (GPUs in this work), where each GPU dispatches each input data to several best-fit experts according to an intelligent gating function and get the corresponding outputs back to combine them. This implies that the workload of experts is fundamentally uncertain – it depends on input data and the gating function. Both of them change at every iteration in practice. In our experiments (see Figure 1), the workload changes up to 4.38× in a single training and different layers have different workload.

Previous DL systems, including the latest MoE frameworks (Lepikhin et al., 2021; Ott et al., 2019; Rajbhandari et al., 2022; He et al., 2022), are mostly based on static runtime execution that does not fit dynamic MoE characteristics. The major pitfall comes from that experts often fail to leverage the best-performing parallelism because the

---

<sup>\*</sup>Equal contribution <sup>1</sup>Microsoft Research <sup>2</sup>Microsoft. Correspondence to: Yongqiang Xiong <yqx@microsoft.com>.

Proceedings of the 6<sup>th</sup> MLSys Conference, Miami Beach, FL, USA, 2023. Copyright 2023 by the author(s).

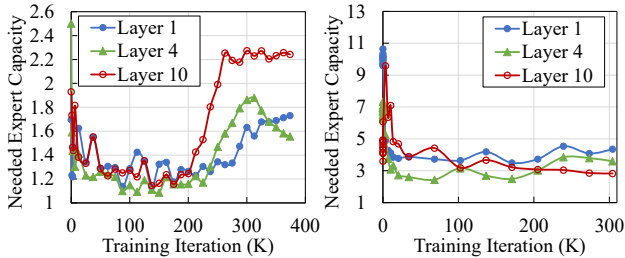


Figure 1. Dynamically changing workload of MoE layers during an end-to-end training of the MoE version of Swin Transformer V2 (Liu et al., 2021; 2022) thin-tiny (left) and base (right) models. The y-axis is the needed expert capacity at runtime, which indicates the amount of workload (see details in Section 2.1). For a neat view, only the 1st, 4th, and 10th layers are shown out of 10 total MoE layers in the model.

optimal one differs depending on the dynamic workload. It is non-trivial to dynamically adjust parallelism at runtime as it typically incurs a large redistribution overhead or GPU memory consumption in existing systems. Other approaches such as *load balancing loss* (Fedus et al., 2022) try to tackle this issue by manipulating the MoE algorithm, but it often harms model accuracy in our experiments (see Section 2.1).

This paper presents TUTEL, a system that thoroughly optimizes MoE at any scale by adaptive methods specialized for dynamic MoE workload. The key mechanism is *adaptive parallelism switching* that dynamically switches the parallelism strategy at every iteration without any extra overhead of switching. Specifically, unlike existing systems that use different tensor layouts for different parallelism strategies, we leverage only a single distribution layout that covers all possibly optimal strategies. This frees the system from reformatting the input data or weights when we switch the parallelism strategy, hence zero-cost switching. Based on our communication cost analysis of all kinds of parallelism, we ensure that adaptive parallelism does not compromise the optimal parallelism strategy.

TUTEL is a fully implemented framework for diverse MoE algorithms at scale. Over the adaptive parallelism switching, it delivers several optimization techniques for efficient and adaptive MoE, including adaptive pipelining, the 2-dimensional hierarchical (2DH) All-to-All algorithm, fast encode/decode with sparse computation on GPU, etc. TUTEL has been open sourced on GitHub<sup>1</sup> and already been integrated into Fairseq (Ott et al., 2019) and DeepSpeed (Microsoft, 2023). Our extensive experiments over Azure A100 clusters (Azure, 2023) show that with 128 GPUs, TUTEL delivers up to 3.11 $\times$  of MoE-layer speedup, and 1.55 $\times$ / 2.11 $\times$  speedup for end-to-end training / inference of a real-world model (SwinV2-MoE), compared to

<sup>1</sup><https://github.com/microsoft/tutel>

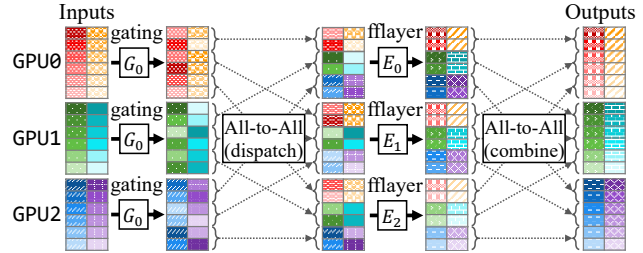


Figure 2. Example of an MoE layer across three GPUs, expert  $E_i$  on GPU  $i$ .  $G_0$  represents the gating function that is shared across all GPUs. Different colors or patterns indicate different samples (columns of inputs) and different gradients of color indicate different tokens within a sample (rows of inputs). This example shows two samples/batch, six tokens/sample, and evenly dispatched top-1 routing with capacity factor 1.0 – see details in Section 2.2.

that of using the original Fairseq. For 2,048 GPUs, the MoE-layer speedup is further improved to 5.75 $\times$ .

Our key contributions are as follows:

- Provide detailed analysis on the dynamic nature of MoE and following challenges in existing frameworks.
- Propose adaptive parallelism switching that efficiently handles dynamic workload of MoE, which achieves 1.35 $\times$   $\sim$  14.57 $\times$  speedup of a single MoE layer.
- Aggregating all acceleration techniques, TUTEL delivers speedup of MoE at any scale: 4.96 $\times$  and 5.75 $\times$  speedup of a single MoE layer over 16 and 2,048 A100 GPUs, respectively.
- TUTEL has been used to implement and run the sparse MoE version of a state-of-the-art vision model, SwinV2-MoE, on real-world computer vision problems. It achieves up to 1.55 $\times$  and 2.11 $\times$  speedup for training and inference, respectively, compared to previous frameworks such as Fairseq. We also demonstrate superior accuracy of the sparse model than the counterpart dense model, indicating the readiness of TUTEL in training real-world AI models.

## 2 BACKGROUND & MOTIVATION

This section introduces the dynamic nature of Mixture-of-Experts and its inefficiency in large-scale training.

### 2.1 Background & Related Work

**Sparsely-gated Mixture-of-Experts (MoE).** MoE employs multiple *expert* models, which deal with their own specialized sub-tasks respectively to solve the entire tasks together. It is leveraged by large-scale distributed DNN models by putting a cross-GPU layer that partially exchanges hidden features from different GPUs (Fedus et al., 2022; Lin et al., 2021; Riquelme et al., 2021). Figure 2 il-

illustrates an example. First, it runs a *gating function* (Lewis et al., 2021; Roller et al., 2021; Yang et al., 2021) that determines the destination GPU of each input token<sup>2</sup> in the following all-to-all collective communication (All-to-All). After the All-to-All (called *dispatch*), each GPU runs their own expert, which is a feed-forward network layer (fflayer), and then conducts the second All-to-All (called *combine*) that sends the corresponding output of each token to the GPU where the token is from. Details of the gating function and the fflayer defer depending on the model algorithm.

**MoE as the Key to Exa-scale Deep Learning.** MoE is differentiated from existing scale-up approaches for DNNs (i.e., increasing the depth or width of DNNs) in terms of its high cost-efficiency. Specifically, enrolling more model parameters (experts) in MoE layers does not increase the computational cost per token. Nowadays, MoE is considered as a key technology for hyper-scale DL with its state-of-the-art results shown in previous works (Fedus et al., 2022; Riquelme et al., 2021; Lepikhin et al., 2021; Du et al., 2022). Currently, many state-of-the-art frameworks (e.g., DeepSpeed (Microsoft, 2023), Fairseq (Ott et al., 2019), etc.) have already supported MoE.

**Dynamic Workload of MoE.** The root cause of dynamic workload of MoE comes from its token routing mechanism. Specifically, MoE layers dynamically route each token to multiple experts, where the distribution of tokens is often uneven across experts. This makes the workload of each expert dynamically change at every iteration as shown in Figure 1. *Expert Capacity* is a common practice to indicate the workload of each expert, which is the number of tokens that an expert receives to deal with. Expert capacity depends on the number of tokens per batch  $T$ , the number of global experts  $E$ , top- $k$  routing ( $1 \leq k \leq E$ ), and the capacity factor  $f$  ( $f \geq 1$ ) as follows:

$$\text{Expert Capacity} = k \cdot f \cdot \frac{T}{E}. \quad (1)$$

$f = 1$  is the minimum value indicating the most even token distribution. A larger  $f$  value indicates more imbalanced token routing, which means that an expert has to deal with more tokens.

Most existing MoE frameworks (Ott et al., 2019; Lepikhin et al., 2021; Rajbhandari et al., 2022; Zheng et al., 2022) simply set  $f$  to a static upper bound of capacity factor  $f_{upper}$  (i.e.,  $f = f_{upper}$ ) so that different iterations always perform a static amount of computation. However, static computation based on  $f_{upper}$  not only introduces unnecessary computations but also may drop excessive to-

<sup>2</sup>Each input sample is divided into one or more tokens, and the definition of a token depends on the model’s algorithm and tasks.

LB Loss Weight	0.001	<b>0.01</b>	<b>0.1</b>	<b>1.0</b>
Acc@1 (%)	37.32	<b>37.78</b>	<b>37.16</b>	<b>34.71</b>

Table 1. Harsh load balancing harms MoE model accuracy. Bold numbers highlight accuracy degradation with large LB loss weights. All experiments are carried on ImageNet-22K image classification and the top-1 accuracy of SwinV2-S model is reported. Hyper-parameters: 32 experts, top-1 routing, capacity factor  $f$ =infinity.

kens from training if  $f_{upper}$  is not set to a sufficiently large value, which potentially impacts the model accuracy. To tackle this, throughout this paper, we consider a system (like TUTEL) that supports MoE training using the minimum required  $f$  that incurs neither unneeded computation nor dropped tokens, as using  $f = f_{upper}$  does. Based on this mechanism, we explore further optimization opportunities while  $f$  varying across training steps.

**MoE Frameworks.** While GShard (Lepikhin et al., 2021) provides a computation logic that ensures algorithmic correctness of MoE, several popular MoE frameworks (Ott et al., 2019; Rajbhandari et al., 2022) follow the same logic but perform poorly on a large scale. Fast/FasterMoE (He et al., 2022) proposes different gating algorithms that are not computationally equivalent with GShard. Furthermore, it proposes *shadow expert* and *smart schedule* that deliver only conditional benefits when imbalanced token distribution persists for a long time, while may harm throughput otherwise. On the other hand, TUTEL pursues keeping the same computation logic as GShard and achieving a deterministic gain over any environments in general, which adapts MoE frameworks to exa-scale without harming algorithmic results.

**Load Balancing Loss.** Load balancing (LB) loss regulates MoE layer training by encouraging gating functions to balance workload of experts (Shazeer et al., 2017; Fedus et al., 2022). LB loss can contribute to low and stable MoE workload as capacity factor  $f$  typically decreases when the token distribution is even (as mentioned in the previous paragraph). However, LB loss is typically insufficient to tackle the dynamic workload of MoE because giving a large weight on the LB loss often harms model accuracy. Specifically, a proper weight on the LB loss may help model accuracy by guiding gating functions to enroll more diverse expert parameters during training, but a too large weight may harm the optimization objectives of the final task, as well as lead to failure of forwarding tokens to their knowledgeable experts. Table 1 shows that our experiments with large LB loss weights harm model accuracy. Additionally, to our empirical findings, LB loss does not always result in more balanced workload across experts. For example, our experiments in Figure 1 use LB loss that

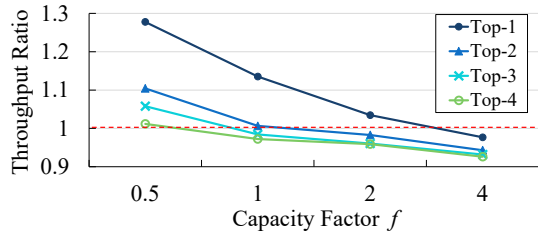


Figure 3. Runtime preferences of two different parallelism methods. The Y-axis measures the throughput ratio of EP+MP to EP+DP. It compares their throughput under varying capacity factor  $f$  (i.e., varying amount of workload) and different top- $k$  configurations, where  $> 1.0$  implies that EP+MP outperforms EP+DP, and vice versa. Model settings: fflayer hidden size 16K, fflayer channel size 2048, and batch size 4.

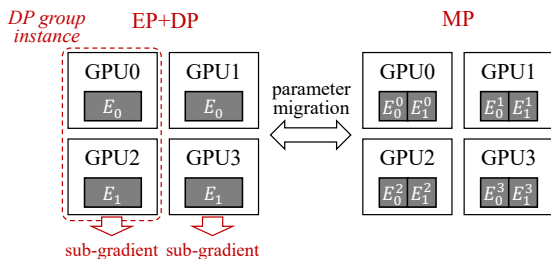


Figure 4. Parameter migration due to switching parallelism between conventional EP+DP and MP.  $E_i^p$  refers to  $p$ -th slice (in the model-parallel manner) of  $i$ -th expert (no  $p$  means not sliced). EP+DP replicates each expert on two GPUs each, and MP slices each expert across four GPUs respectively.

help achieve the best accuracy, but it still shows dynamically changing workload. In this paper, we only consider system-side solutions that are generally applied regardless of the LB loss.

## 2.2 Static Parallelism

Under the dynamic nature of MoE layers, it becomes challenging if we would like to accelerate one expert with multiple GPUs for higher throughput. Previous research has proven that employing more experts typically gains only fast diminishing incremental benefits with many experts ( $> 256$ ) (Rajbhandari et al., 2022; Clark et al., 2022; Fedus et al., 2022). Therefore, in large-scale training, MoE layers typically employ relatively small number of experts compared with the number of GPUs and multiple GPUs are assigned to one expert for higher throughput.

We consider three different parallelism methods that have been adopted for MoE in prior works (Fedus et al., 2022): expert parallelism (EP, distribute experts), data parallelism (DP, distribute input data), and model parallelism (MP, split and distribute a single expert). EP, DP, and MP can be used at the same time with each others.

Number of GPUs	16	64	256
MoE overhead (ms)	560.9	698.9	866.4
Computation overhead (ms)	371.8	375.1	386.3
All-to-All overhead (ms)	189.1	323.8	491.3
All-to-All overhead ratio	33.7%	46.3%	56.7%
Potential overhead saving	33.7%	46.3%	43.3%
<b>Potential speedup</b>	<b>1.51<math>\times</math></b>	<b>1.86<math>\times</math></b>	<b>1.76<math>\times</math></b>

Table 2. Ratio of All-to-All overhead and potential speedup by fully overlapping All-to-All and computation in a typical MoE setting. Model settings: fflayer hidden size 4K, fflayer channel size 4K, 2 experts per GPU, 64K tokens per iteration.

According to our experiments, statically adopting a certain parallelism method does not always work efficiently under dynamic workload. For example, Figure 3 compares performance of two different parallelism methods, EP+DP and EP+MP. As shown in the figure, the best parallelism method depends on the workload, which has 7.39%-27.76% performance gap between these two parallelisms.

Unfortunately, switching between different parallelism methods during runtime would incur a substantial overhead. Specifically, in existing work, an on-going training based on a certain parallelism (e.g., data-parallel) is not designed to be compatible with another parallelism (e.g., model-parallel) because they have different requirements on data split, weight split, managing momentum of parameter gradients, and even the framework interfaces to launch the training. Furthermore, parameter migration is another costly overhead that would be incurred when we change the parallelism, as illustrated in Figure 4. These are why parallelism switching is hardly used in existing systems.

## 2.3 Static Pipelining

MoE layers shown in Figure 2 often under-utilize GPUs as they run All-to-All and fflayer in sequence to dispatch and combine. As All-to-All mostly consists of inter-GPU data copies that are not compute-intensive, we can better utilize computational power of GPUs by pipelining it with fflayer that runs numeric computation. Table 2 shows up to 1.86 $\times$  potential speedup by overlapping All-to-All and fflayer computation.

However, we observe that the static pipelining strategy for dispatch and combine, namely static All-to-All algorithm and pipelining degree, are inefficient to handle the dynamic workload. As illustrated in Figure 5, depending on different MoE settings and scales, the corresponding optimal pipelining strategy consists of various All-to-All algorithms (*Linear* or *2DH*<sup>3</sup>) and pipelining degrees. This

<sup>3</sup>While Linear All-to-All lets all GPUs directly communicate with each others, 2DH (2-Dimensional Hierarchical) All-to-All adopts a hierarchical algorithm that conducts intra-node communication in a separate earlier stage. 2DH tends to outperform Lin-

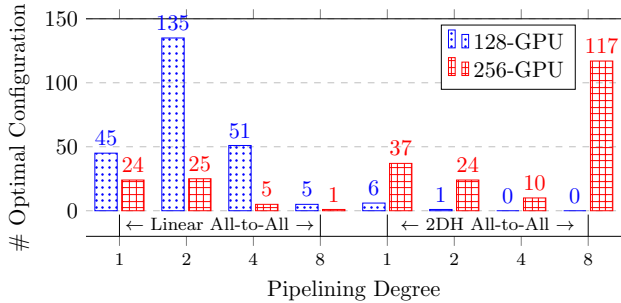


Figure 5. The distribution of optimal pipeline strategies for various MoE workload configurations. Each column indicates the number of configurations that perform best with the strategy described on X-axis. Details of workload configurations are the same as described in Section 5.1.2.

Symbol	Description
$W$	World size used for All-to-All exchange
$D$	fflayer channel size for each sample
$H$	fflayer hidden size for each sample
$E_g$	Number of local experts per GPU
$E$	Number of global experts
$C_g$	Token capacity per GPU
$C$	The total token capacity across GPUs
$P$	The total parameters of all experts
$f$	The capacity factor used in Equation (1)

Table 3. Description of symbols.

means that a single static strategy cannot always achieve the optimal performance in different MoE settings and scales, and dynamic pipelining strategy is necessary at runtime to adapt to varying settings.

To make things worse, the interference between computation and communication makes it difficult to find the optimal pipelining strategy if we only consider each single aspect separately. This is because the slowdown from running NCCL kernels concurrently with computation kernels on the same GPU is difficult to estimate. To our extensive experiments, even when two different All-to-All algorithms have similar throughputs, their throughputs often differ a lot when the same concurrent computation kernel is introduced, and either algorithm may outperform another one case-by-case. This implies that the dynamic adjustment should be done jointly with both computation and communication for the optimal overall throughput.

### 3 ADAPTIVE MOE WITH TUTEL

TUTEL, a full-stack MoE system, supports a complete MoE layer with adaptive optimizations. As all optimizations are transparent to DNN model developers, TUTEL would not

change the interface of DL frameworks and it can easily be integrated with other frameworks. In the following subsections, we describe how TUTEL tackles the aforementioned problems in detail.

#### 3.1 Adaptive Parallelism Switching

##### 3.1.1 What is the least subset that is deserved for Parallelism Switching?

Given that EP, DP, and MP derive 7 different possible combinations of parallelism methods, an ad-hoc approach is to design one execution flow for each method and makes it switchable with all other methods. However, designing up to 7 execution flows is not necessary as the problem can be precisely simplified into a smaller but *efficiency-equivalent* problem, as is highlighted in the subsection title.

Our approach is analyzing complexity of all parallelism methods to narrow them down to the least subset that we need to design execution flows for. Note that only communication complexity matters here because all GPUs conduct an identical computation, hence the same computational complexity, so the communication complexities directly determines the efficiency of one parallelism method against others. As shown in Table 4, we analyze communication complexities of all parallelism methods to remove those from our consideration if they are (1) not the optimal in any cases or (2) a special case of another method. By a series of comparison (shown in the Comment column of Table 4), we draw a conclusion that the subset can include only DP and EP+DP+MP. Therefore, the following paragraphs design corresponding parallel structure focusing only on DP and EP+DP+MP, which still guarantees to cover the optimal parallelism method regardless of model configurations.

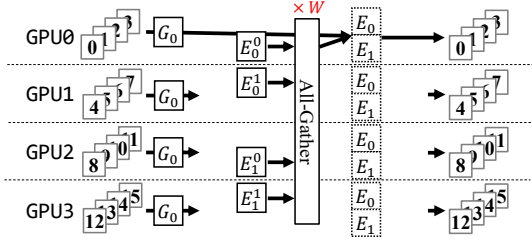
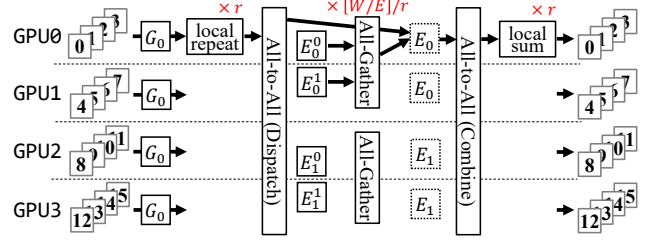
##### 3.1.2 Execution Flow of Zero Cost Switchable Parallelism

As explained in Section 2.2, the switchable parallelism should guarantee exactly the same data layout and execution flow of MoE training. We explain our design for DP and EP+DP+MP respectively as follows. Zero Cost means that switching parallelism is completely free, without introducing any overhead larger than  $\mathcal{O}(1)$  from parameter/token migration.

**Switchable DP (Figure 6):** It follows the conventional DP training that takes only local tokens as input, but weight parameters following the ZeRO-DP Stage-3 Partitioning (Rajbhandari et al., 2020) mechanism. Specifically, it lets each device to own a unique slice of weights, and performs one all-gather communication during the forward-pass and one reduce-scatter communication during the backward-pass, instead of the conventional training that performs one all-

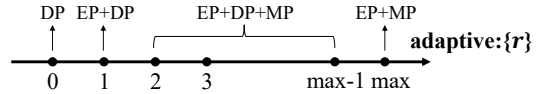
Parallelism Method	Communication Complexity	Limitation	Comment
① DP	$\mathcal{O}(P)$	-	Possibly optimal
② MP	$\mathcal{O}(C_g \cdot W)$	-	No better than ⑥
③ EP	$\mathcal{O}(C_g)$	$E/W \geq 1$	No better than ⑥
④ DP+MP	$\mathcal{O}(C_g \cdot r + P/r)$	$1 \leq r \leq W$	No better than ⑦ for any $r$
⑤ EP+DP	$\mathcal{O}(C_g + P/E)$	-	A special case of $r = 1$ in ⑦
⑥ EP+MP	$\mathcal{O}(C_g \cdot \max\{1, W/E\})$	-	A special case of $r = W/E$ in ⑦
⑦ EP+DP+MP	$\mathcal{O}(C_g \cdot W/E)$ – if $r \geq W/E$ $\mathcal{O}(C_g \cdot r + P/E/r)$ – if $1 \leq r < W/E$	-	Possibly optimal

Table 4. Analysis on communication complexity of MoE parallelism.


 Figure 6. An example of DP execution flow in TUTEL. All-gather is performed across all ( $W$ ) GPUs.

 Figure 7. An example of EP+DP+MP execution flow in TUTEL. Local repeat generates  $r$  copies of gating function results, local sum reduces  $r$  outputs from MoE combine, and all-gather is performed across  $\lceil (W/E)/r \rceil$  GPUs.

reduce communication during the backward-pass. Both ways are complexity-equivalent as a single all-reduce naturally consists of a reduce-scatter and an all-gather. In Figure 8,  $r = 0$  stands for the Switchable DP.

**Switchable EP+DP+MP (Figure 7):** Out of the box, this parallelism method works the same as the Switchable DP – they share the same format of reading inputs and slicing weights. Within the box, it not only ensures that the whole computation is mathematically equivalent to DP, but also ensures the required computation and network complexity are within the expected complexity of EP+DP+MP as shown in ⑦ of Table 4. We define a control parameter  $r$  that indicates to partition all GPUs into one or more groups with size  $\lceil (W/E)/r \rceil$  each, so that DP will be performed within each group and MP will be performed across different groups. Specifically, it repeats local tokens  $r$  in the style of MP at the beginning of execution flow, and finally performs a local sum symmetrically in the end. DP is only used to perform all-gather within a group of size  $\lceil (W/E)/r \rceil$ . Note that if  $r$  increases and reaches  $W/E$ , the group size becomes 1, thus all-gather communication within each group is optimized out. This is why the case  $r \geq W/E$  in ⑦ eliminates an additional  $\mathcal{O}(P/E/r)$ . In Figure 8,  $r$  values from 1 to  $W/E$  stands for the Switchable EP+DP+MP, though  $r = 1$  and  $r = \lceil W/E \rceil$  are two special cases that are exactly equivalent with EP+DP and EP+MP respectively.


 Figure 8. Specifying a parallelism method using `adaptive:r`, with `max` standing for the value  $\lceil W/E \rceil$ , and all  $r$  values larger than this upper-bound are regarded the same as  $\lceil W/E \rceil$ .

### 3.2 Adaptive Pipelining for Linear & 2DH All-to-All

This section presents the design of adaptive pipelining. As All-to-All communication latency substantially impacts on the optimal pipelining degree, our adaptive pipelining jointly optimizes both pipelining degree and All-to-All algorithms (Linear or 2DH) at the same time. While this section only explains how to partition input tokens for pipelining, the following Section 3.3 describes how we jointly search for the optimal pipelining degree and the All-to-All communication algorithm.

**Token partition for multi-stream pipelining.** Tokens need to be partitioned properly to enable the overlapping of flows on finer-grained data chunks, so that computation and communication can be submitted on separate GPU streams and run in parallel. Traditional partitioning like batch-splitting or pipeline-parallelism (Huang et al., 2019) partitions all operations in the layer. This doesn't work in MoE because it amplifies the imbalance of MoE dispatch

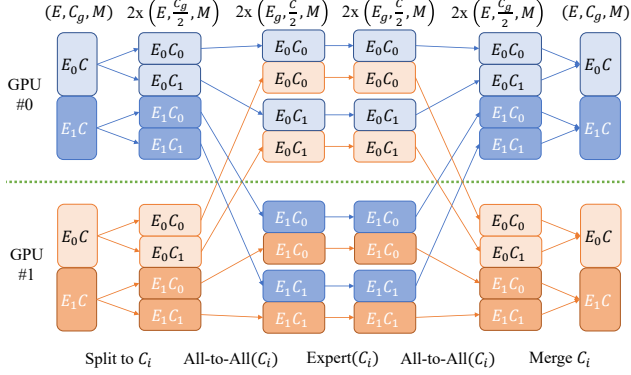


Figure 9. Overview of token partition on 2-expert-2-GPU for All-to-All-Expert multi-stream overlapping.  $E_i$  means data is sent to  $i$ -th GPU and processed by  $i$ -th expert, and  $C_i$  means data belongs to  $i$ -th partition of capacity dimension. All-to-All and expert operations of different capacity partitions can be overlapped.

and destroys correctness for ML features like Batch Prioritized Routing (Riquelme et al., 2021). Instead, we propose to only partition the two All-to-Alls and the expert in between instead of the whole MoE layer to avoid those shortcomings. Figure 9 gives 2-GPU example for data partition design in All-to-All-Expert overlapping.

In the forward pass, on each GPU, input of shape  $(E, C_g, D)$  is split along dimension  $C$  into two virtual partitions of shape  $(E, C_g/2, D)$ . These two virtual partitions are marked with  $C_0$  and  $C_1$ . After the splitting, each virtual partition  $C_i$  is asynchronously sent to execute All-to-All operation in  $i$ 's order, on communication stream. All-to-All is customized to accept segregated data chunks as input and perform inline data shuffling, generating output of shape  $(E_g, C/2, D)$ . Next, the two All-to-All outputs are programmed to be sent to execute expert computation on computation stream once their previous corresponding All-to-All is completed, and the outputs of expert computation are again programmed to be sent to execute the second All-to-All on communication stream once previous corresponding expert computation is completed. Finally, a barrier is set after the second All-to-Alls, After the barrier, partitions are merged to generate final output of shape  $(E, C_g, D)$ .

The backward pass works in a similar way as the forward pass, except that the input becomes the gradients of the original output, the computation becomes the backward computation of the expert, and the output becomes the gradients of the original input.

Note that all partitioning and reshaping operations are done inline by customized operations, hence no extra data copy overhead compared with no-overlapping cases.

### 3.3 Dictionary of Optimal Parallelism & Pipelining

TUTEL manages a dictionary to memorize the optimal parallelism and pipelining setup of various different ranges of expert capacities. Specifically, we define the dictionary as a hash map:  $\lfloor c/R \rfloor \rightarrow \{r^*, d^*, a^*\}$ , where  $c$  is a capacity value of a certain iteration,  $R$  is the window size that converges multiple adjacent  $c$  values into the same key (default is 128), and  $\{r^*, d^*, a^*\}$  is a tuple of the optimal setup (adaptive:  $r$ , pipelining degree, and All-to-All algorithm, respectively). To build up this dictionary beforehand, we need to find the optimal setup of each possible key ( $\lfloor c/R \rfloor$ ) that only requires a few trials, which is calculated as:

$$\text{trials per key} = (\log_{3/2} \lceil W/E \rceil + 2) \cdot 4 \cdot 2.$$

$(\log_{3/2} \lceil W/E \rceil + 2)$  is the number of needed trials to search for  $r^*$  via Ternary Search (Wikipedia, 2023) because  $r$  in range  $[1, \lceil W/E \rceil - 1]$  determines a convex optimal distribution, plus two extra trials for  $r = 0$  and  $r = \lceil W/E \rceil$ . “4” is the number of needed trials for  $d^*$  as we limit the search space of the pipelining degree to  $\{1, 2, 4, 8\}$ . To our practices, larger degrees than 8 hardly improve the overlapping between computation and communication, while significantly inflating All-to-All overhead. “2” refers to the number of All-to-All algorithms (Linear or 2DH).

## 4 IMPLEMENTATION

### 4.1 Features

TUTEL provides more comprehensive support on MoE model training for different devices, data types and MoE-related features compared with other MoE frameworks, including DeepSpeed MoE, Fairseq MoE, and FastMoE.

**Dynamic Top-ANY MoE Gating.** To enable a variety of sparsity options for MoE training, TUTEL supports top-ANY routing. The  $k$  value can be customized per step as well to enable dynamic sparsity updates, which is useful when different iterations of one MoE layer use their preferred top- $k$  settings instead of using the same  $k$  value. Users can leverage this feature to dynamically fine-tune sparsity of MoE layers.

**Dynamic Capacity Factor.** To smartly control the capacity upper-bound under varying token imbalance, TUTEL supports adjusting the capacity factor dynamically at every iterations. As illustrated in Figure 10, the adjustment behavior is controlled by argument `capacity_setting = x` passed to our MoE layer API. If  $x$  is positive, the value is directly applied as the capacity factor of the MoE layer. If  $x$  is zero, TUTEL automatically adapts the capacity factor to the minimum value that does not drop any tokens at each iteration. If  $x$  is negative, it works the same as when  $x$  is

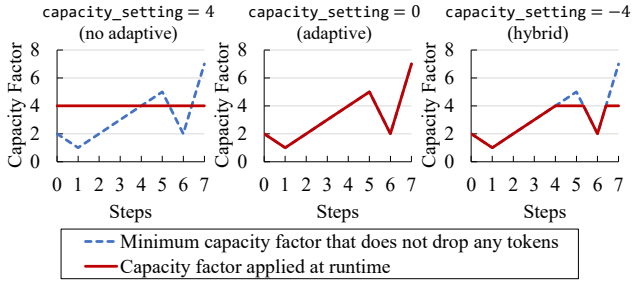


Figure 10. Examples of dynamic capacity factor adaptation when `capacity_setting` is given as 4, 0, and  $-4$ , respectively.

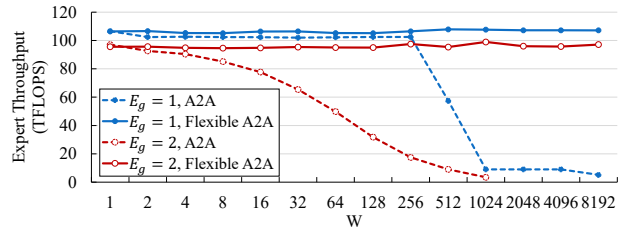


Figure 11. Throughput for expert computation based on A2A (All-to-All) layout and Flexible A2A layout.

zero except that  $-x$  is set as the upper bound of capacity factor, i.e., any exceeding value will be adapted to  $-x$ .

### 4.2 Optimizations

**Flexible All-to-All.** We propose an abstraction upon conventional MPI/NCCL All-to-All interfaces to ensure high computational throughput of MoE experts regardless of the scale, which is called *Flexible All-to-All* in this context. Existing All-to-All transforms the tensor layout from  $(E, C_g, D)$  into  $(W, E_g, C_g, D)$  where  $C_g$  relies on  $W$ , which affects the efficiency of the following matrix multiplication by experts. Instead, we transform the output layout into  $(E_g, C, D)$  that ensures the same-shaped matrix multiplication at any scale  $(W)$ . Figure 11 compares the expert computation throughput between the conventional All-to-All and Flexible All-to-All.

**Kernel Optimization: Fast Encode and Decode.** According to GShard (Lepikhin et al., 2021), existing implementations for MoE dispatch and combine need multiple einsum and matrix multiplication operations. TUTEL deeply optimizes this by using SIMT-efficient sparse operations, which we call *fast encode and decode*. It largely minimizes the latency of non-expert computations, as shown in Figure 15. This optimization saves GPU memory as well, achieving 20% ~ 90% memory saving in most cases. See more details of fast encode and decode in Appendix B.

tokens/step	Fairseq MoE (GiB)	TUTEL MoE (GiB)
4,096	3.7	2.9 (-21.6%)
8,192	6.2	3.2 (-48.4%)
16,384	16.3	4.0 (-75.5%)
32,768	57.9	5.7 (-90.2%)

Table 5. GPU memory cost for single MoE layer. (Static Settings:  $D = H = 4096$ , top-k = 2,  $E_g = 2$ )

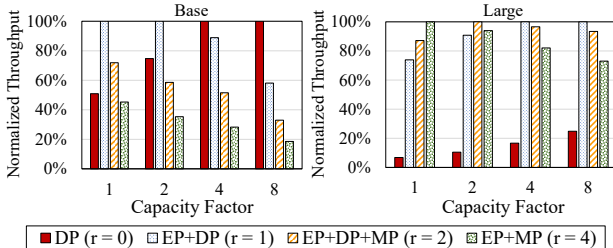


Figure 12. Normalized throughputs of different capacity factors  $f$  under Base (left) and Large (right) MoE configurations. The figure shows that the optimal parallelism method differs depending on capacity factor  $f$ .

## 5 EVALUATION

**Testbed.** If not specified, all experiments use Azure Standard\_ND96amsr\_A100\_v4 VMs (Azure, 2023). Each VM is equipped with  $8 \times$  NVIDIA A100 SXM 80GB GPUs and  $8 \times$  200 Gbps HDR InfiniBand, backed by  $96 \times$  2nd-generation AMD Epyc CPU cores and 1.9 TiB memory. GPUs are connected by 3rd-generation NVLink and NVSwitch within one VM, while different VMs are connected through 1,600 Gbps InfiniBand non-blocking network with adaptive routing.

**Setup.** For baseline, we use PyTorch 1.8.0 and Fairseq moe branch by default. NCCL 2.10.3-1 (NVIDIA, 2023a) and NCCL RDMA SHARP plugin (Mellanox, 2023) are used for communication when scaling out. We use up to 2,048 A100 GPUs (256 VMs) for experiments.

### 5.1 Evaluation on Adaptive MoE with TUTEL

This section evaluates gains from adaptive computation using TUTEL. We compare the throughput of optimal parallelism / pipelining strategy and study the gain from adaptivity of TUTEL. For apples-to-apples comparison with existing frameworks, in Section 5.2, we compare TUTEL with Fairseq MoE (Ott et al., 2019) only using a specific parallelism method that is supported by both.

#### 5.1.1 Adaptive Parallelism Switching

We evaluate adaptive parallelism switching with various MoE model settings using a single node. Figure 12 compares normalized throughputs using different parallelism options where capacity factor  $f$  varies from 1.0 to 8.0. We test two MoE configurations, Base ( $samples/step = 4K$  and  $H = 2K$ ) and Large ( $samples/step = 1K$  and



GPU Num.	All2All Algo.	Pipelining Degree			
		1	2	4	8
16	Linear	20%	2%	2%	11%
	2DH	101%	98%	100%	106%
32	Linear	16%	1%	2%	11%
	2DH	45%	43%	44%	51%
64	Linear	13%	1%	5%	15%
	2DH	28%	25%	27%	34%
128	Linear	9%	2%	9%	29%
	2DH	16%	16%	19%	26%
256	Linear	20%	27%	54%	107%
	2DH	12%	20%	34%	11%

(a) Adaptive pipelining improvement on average.

GPU Num.	All2All Algo.	Pipelining Degree			
		1	2	4	8
16	Linear	60%	32%	50%	176%
	2DH	149%	139%	142%	184%
32	Linear	60%	31%	41%	135%
	2DH	89%	75%	59%	148%
64	Linear	55%	23%	42%	161%
	2DH	70%	54%	41%	109%
128	Linear	45%	54%	87%	300%
	2DH	52%	37%	35%	107%
256	Linear	100%	160%	317%	599%
	2DH	73%	139%	193%	182%

(b) Adaptive pipelining improvement over the worst case.

Table 6. Adaptive pipelining improvements.

$H = 32K$ ), while other expert settings are shared ( $E = 16$ ,  $D = 2K$ , and 64 total GPUs). As shown in the figure, the optimal parallelism method varies depending on the MoE expert configurations and capacity configurations. For instance, DP ( $r = 0$ ) tends to be more favorable when the expert capacity is high, and as the capacity decreases, the tendency gradually changes to EP+DP ( $r = 1$ ) and then to EP+DP+MP ( $r > 1$ ). In relatively smaller-scale MoE configurations, the optimal parallelism option typically stays in  $r = 0$  or  $r = 1$ , while it dynamically changes across a wider range of  $r$  values in larger-scale configurations. Such a variety evidences a substantial chance of improvements with TUTEL, which leads to different optimal parallelism methods according to the dynamically changing  $f$ , as explained in Section 3.1.

### 5.1.2 Adaptive Pipelining

We evaluate adaptive pipelining on 243 typical MoE model settings on different scale (16 ~ 256 GPUs). We test all combinations of MoE model configurations within:  $E_g \in \{0.5, 1, 2\}$ ,  $D \in \{1024, 2048, 4096\}$ ,  $H \in \{1024, 2048, 4096\}$ , and tokens/step  $\in \{4096, 16384, 65536\}$ . For comparison, we also measure different static pipelining methods considering different degrees  $\{1, 2, 4, 8\}$  and different All-to-All algorithms (Linear or 2DH).

Table 6a shows average improvement on these 243 models. Compared with baseline solution (pipelining degree 1) and Linear All-to-All, adaptive pipelining achieves 9% ~

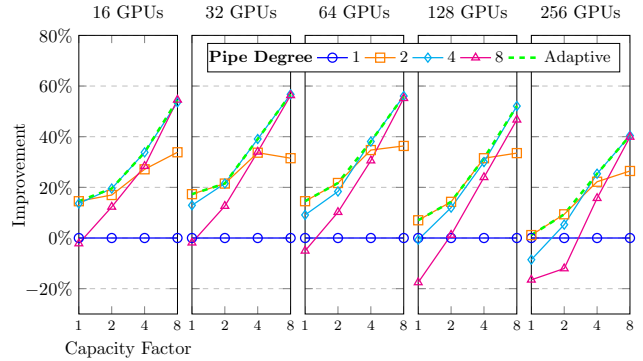


Figure 13. Improvement from adaptive pipelining depending on capacity factor  $f$ .  $D = 4096$ ,  $H = 4096$ ,  $E_g = 2$ , and tokens/step = 4096.

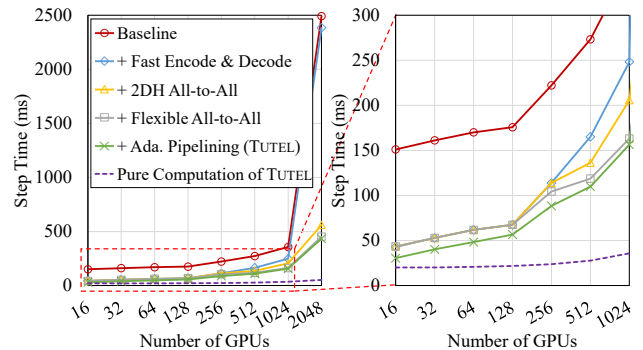


Figure 14. Single MoE layer improvement breakdown. The baseline is a Fairseq / DeepSpeed MoE layer.

101% improvement in average. Compared with different static strategies, it also can achieve 1% ~ 107% improvement in average. Besides, adaptive pipelining achieves significant improvement and avoids performance regression in the worst case, which shows 23% ~ 599% improvement in Table 6b.

We also evaluate the performance gain under different dynamic workloads on different scales. We use different capacity factors  $f$  to emulate different workload patterns in different training iterations. As shown in Figure 13, adaptive pipelining always chooses the best strategy, and it can achieve up to 39% improvement with  $f = 4$  and up to 57% improvement with  $f = 8$ , compared with baseline (pipelining degree 1).

### 5.2 Single MoE Layer Scaling

We evaluate the step time of single MoE layer when scaling out to 2,048 GPUs. It uses tokens/step = 16384,  $f = 1$ ,  $D = 2048$ ,  $H = 2048$ ,  $E_g = 2$ , top- $k = 2$ , adaptive: $r = 1$ . We add TUTEL features once at a time to study where the major gain is from, where Fairseq (Ott et al., 2019) is used as the baseline. Detailed experiments for each feature are provided in the following Section 5.1.

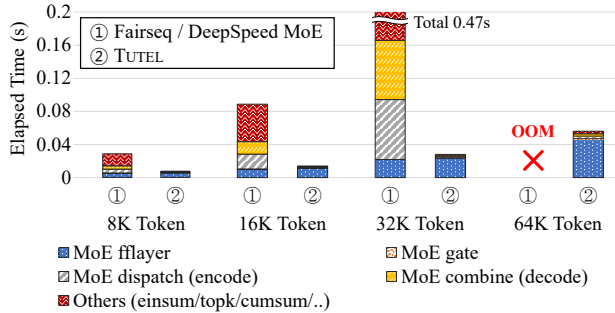


Figure 15. Kernel computation breakdown comparison between TUTEL and Fairseq / DeepSpeed MoE.

The following explains each curve in Figure 14 in order. ① (red, circle) Fairseq / DeepSpeed MoE Baseline. Fairseq and DeepSpeed MoE perform the same as they use an equivalent MoE layer implementation. ② (blue, diamond) TUTEL Kernel (Fast Encode & Decode in Section 4.2) + Linear All-to-All. TUTEL kernel optimizations deliver a large gain at a small scale ( $3.52\times$  on 16 GPUs), while the gain becomes small at a large scale ( $1.04\times$  on 2,048 GPUs). The detailed gains from using TUTEL kernels over Fairseq are shown in Figure 15. ③ (yellow, triangle) TUTEL Kernel + 2DH All-to-All. 2DH All-to-All delivers a significant gain on a large scale ( $4.25\times$  on 2,048 GPUs). ④ (gray, square) TUTEL Kernel + 2DH All-to-All + Flexible All-to-All. Flexible All-to-All delivers gains on large scales starting from 256 GPUs, e.g.,  $1.24\times$  on 2,048 GPUs compared with not using it. ⑤ (green, cross) TUTEL Kernel + 2DH All-to-All + Flexible All-to-All + Adaptive Pipelining Degree. ⑤ shows the mixture of gains from optimizing the pipelining degree together with Linear/2DH All-to-All algorithms, further achieving  $1.43\times$  and  $1.04\times$  on 16 and 2,048 GPUs, respectively. ⑤ becomes less important on larger scales as the overhead of slicing tokens becomes more detrimental to All-to-All efficiency. The breakdown does not include adaptive parallelism switching as it statically uses  $\text{adaptive}:r = 1$ , not only because this parallelism is officially supported by Fairseq MoE while others are not, but also to ensure that the All-to-All communication size required by TUTEL and Fairseq MoE are exactly the same, so as to fairly compare the improvement of All-to-All.

Compared with the baseline, TUTEL finally delivers  $4.96\times$ ,  $3.11\times$ , and  $5.75\times$  speedup on 16 GPUs, 128 GPUs, and 2,048 GPUs, respectively. For computation-communication breakdown, ⑥ (purple, dashed) shows the pure computation overhead of the complete TUTEL (excluding the portion overlapped with communication). Note that the slight increase of computation overhead as we scale out is not from the system overhead but due to more theoretical computation required by the gating function for total  $E_g \cdot W$  experts.

#GPU	Dense train / infer	Fairseq MoE train / infer	TUTEL MoE train / infer	Speedup train / infer
8	291 / 1198	240 / 507	274 / 1053	$1.14\times / 2.08\times$
16	290 / 1198	173 / 473	253 / 943	$1.46\times / 1.99\times$
32	288 / 1195	162 / 455	249 / 892	$1.54\times / 1.96\times$
64	285 / 1187	159 / 429	234 / 835	$1.47\times / 1.95\times$
128	256 / 1103	146 / 375	226 / 792	$1.55\times / 2.11\times$

Table 7. Comparing the training and inference speeds (images per second) of SwinV2-MoE using Fairseq and TUTEL.

### 5.3 Adoption to Real-world Problems: SwinV2-MoE

We introduce SwinV2-MoE to verify the correctness and performance of TUTEL in end-to-end training and testing. SwinV2-MoE is an MoE version of Swin Transformer V2 (Liu et al., 2021; 2022), which is a state-of-the-art computer vision neural network architecture that is widely used in a large variety of computer vision problems. SwinV2-MoE is built from a dense Swin Transformer V2 model with every other feed-forward layer replaced by an MoE layer except for the first two network stages. The SwinV2-B model is adapted for experiments, and the default hyperparameters are:  $E = 32$ ,  $\text{top-}k = 1$ , and  $f = 1.0$ .

#### 5.3.1 Experiment Setup

##### Pre-training and Down-stream Computer Vision Tasks.

We follow (Liu et al., 2021) to use ImageNet-22K image classification datasets for model pre-training, which contains 14.2 million images and 22 thousand classes. In addition to evaluating the performance of the pre-training task (using a validation set with each class containing 10 randomly selected images), we also evaluated the models using 3 down-stream tasks: 1) ImageNet-1K fine-tuning accuracy. The pre-trained models are fine-tuned on ImageNet-1K training data and the top-1 accuracy on the validation set is reported; 2) ImageNet-1K 5-shot linear evaluation (Riquelme et al., 2021). 5 randomly selected training images are used to train a linear classifier, and the top-1 accuracy on the validation set is reported; 3) COCO object detection (Lin et al., 2014). The pre-trained models are fine-tuned on the COCO object detection training set using a cascade mask R-CNN framework (Liu et al., 2021), and box/mask AP on the validation set is reported.

#### 5.3.2 Experiment Results

**Speed Comparison.** Table 7 compares the training and inference speeds of SwinV2-MoE using Fairseq and TUTEL. For all GPU numbers, from 8 to 128 (1 expert per GPU), TUTEL is significantly faster than Fairseq in both training and inference. Speedup of each iteration is  $1.14\times \sim 1.55\times$  and  $1.95\times \sim 2.11\times$  in training and inference, respectively.

Method	IN-22K	IN-1K/ft	IN-1K/5-shot	COCO (AP)
	acc@1	acc@1	acc@1	box / mask
SwinV2-B	37.2	85.1	75.9	53.0 / 45.8
SwinV2-MoE-B	38.5	85.5	77.9	53.4 / 46.2

Table 8. Comparing the pre-training and fine-tuning accuracy between the sparse SwinV2-MoE-B model and its dense counterpart SwinV2-B.

**Accuracy Comparison.** We report the results of SwinV2-MoE-B on both pre-training and down-stream tasks, compared to the counterpart dense models, as shown in Table 8. SwinV2-MoE-B achieves a top-1 accuracy of 38.5% on the ImageNet-22K pre-training task, which is +1.3% higher than the counterpart dense model. It also achieves higher accuracy on down-stream tasks: 85.5% top-1 accuracy on ImageNet-1K image classification, 77.9% top-1 accuracy on 5-shot ImageNet-1K classification, and 53.4/46.2 box/mask AP on COCO object detection, which is +0.4%, +2.0%, and +0.4/+0.4 box/mask AP higher than that using dense modes, respectively. In particular, it is the first time that the sparse MoE model is applied and demonstrated beneficial on the important down-stream vision task of COCO object detection.

## 6 CONCLUSION

In this paper, we analyze the key *dynamic* characteristics in MoE from system’s perspectives. We address consequent issues by designing an *adaptive* system for MoE, TUTEL, which we present in two major aspects: adaptive parallelism for optimal expert execution and adaptive pipelining for tackling inefficient and non-scalable dispatch/combine operations in MoE layers. We evaluate TUTEL in an Azure A100 cluster with 2,048 GPUs and show that it achieves up to  $5.75\times$  speedup for a single MoE layer. TUTEL empowers both training and inference of real-world state-of-the-art deep learning models. As an example, this paper introduces our practice that adopts TUTEL for developing SwinV2-MoE, which shows effectiveness of MoE in computer vision tasks comparing against the counterpart dense model.

## ACKNOWLEDGEMENTS

We appreciate the feedback by our shepherd, Lianmin Zheng, as well as anonymous reviewers of MLSys’23.

## REFERENCES

Azure, M. NDm A100 v4-series - Azure Virtual Machines. <https://docs.microsoft.com/en-us/azure/virtual-machines/ndm-a100-v4-series>, 2023. [Online; accessed Apr 2023].

Bruck, J., Ho, C.-T., Kipnis, S., Upfal, E., and Weathersby, D. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Transactions on parallel and distributed systems*, 8(11):1143–1156, 1997.

Chi, Z., Dong, L., Huang, S., Dai, D., Ma, S., Patra, B., Singhal, S., Bajaj, P., Song, X., and Wei, F. On the representation collapse of sparse mixture of experts. *CoRR*, abs/2204.09179, 2022.

Clark, A., de Las Casas, D., Guy, A., Mensch, A., Paganini, M., Hoffmann, J., Damoc, B., Hechtman, B. A., Cai, T., Borgeaud, S., van den Driessche, G., Rutherford, E., Hennigan, T., Johnson, M. J., Cassirer, A., Jones, C., Buchatskaya, E., Budden, D., Sifre, L., Osindero, S., Vinyals, O., Ranzato, M., Rae, J. W., Elsen, E., Kavukcuoglu, K., and Simonyan, K. Unified scaling laws for routed language models. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2022.

Cowan, M., Maleki, S., Musuvathi, M., Saarikivi, O., and Xiong, Y. MSCCLang: Microsoft collective communication language. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023.

Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A. W., Firat, O., Zoph, B., Fedus, L., Bosma, M. P., Zhou, Z., Wang, T., Wang, Y. E., Webster, K., Pellat, M., Robinson, K., Meier-Hellstern, K. S., Duke, T., Dixon, L., Zhang, K., Le, Q. V., Wu, Y., Chen, Z., and Cui, C. Glam: Efficient scaling of language models with mixture-of-experts. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2022.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

He, J., Zhai, J., Antunes, T., Wang, H., Luo, F., Shi, S., and Li, Q. Fastermoe: Modeling and optimizing training of large-scale dynamic pre-trained models. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2022.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M. X., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., and Chen, Z. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

- Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020.
- Kim, J., Dally, W. J., Scott, S., and Abts, D. Technology-driven, highly-scalable dragonfly topology. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. IEEE, 2008.
- Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. Gshard: Scaling giant models with conditional computation and automatic sharding. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.
- Lewis, M., Bhosale, S., Dettmers, T., Goyal, N., and Zettlemoyer, L. BASE layers: Simplifying training of large, sparse models. In Meila, M. and Zhang, T. (eds.), *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.
- Lin, J., Yang, A., Bai, J., Zhou, C., Jiang, L., Jia, X., Wang, A., Zhang, J., Li, Y., Lin, W., Zhou, J., and Yang, H. M6-10T: A sharing-delinking paradigm for efficient multi-trillion parameter pretraining. *CoRR*, abs/2110.03888, 2021.
- Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.
- Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.
- Liu, Z., Hu, H., Lin, Y., Yao, Z., Xie, Z., Wei, Y., Ning, J., Cao, Y., Zhang, Z., Dong, L., Wei, F., and Guo, B. Swin transformer v2: Scaling up capacity and resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.
- Mellanox. RDMA and SHARP Plugins for NCCL Library. <https://github.com/Mellanox/nccl-rdma-sharp-plugins>, 2023. [Online; accessed Apr 2023].
- Microsoft. DeepSpeed. <https://www.deepspeed.ai/>, 2023. [Online; accessed Apr 2023].
- NVIDIA. What is LL128 Protocol? <https://github.com/NVIDIA/nccl/issues/281>, 2020a. [Online; accessed Apr 2023].
- NVIDIA. NVIDIA A100 Tensor Core GPU Architecture – Unprecedented Acceleration at Every Scale. Whitepaper, 2020b.
- NVIDIA. NVIDIA Collective Communications Library (NCCL). <https://github.com/NVIDIA/nccl/tree/v2.10.3-1>, 2023a. [Online; accessed Apr 2023].
- NVIDIA. Point-to-point communication – NCCL 2.10.3 documentation. [https://docs.nvidia.com/deeplearning/nccl/archives/nccl\\_2103/user-guide/docs/usage/p2p.html](https://docs.nvidia.com/deeplearning/nccl/archives/nccl_2103/user-guide/docs/usage/p2p.html), 2023b. [Online; accessed Apr 2023].
- NVIDIA. NCCL Tests. <https://github.com/NVIDIA/nccl-tests>, 2023c. [Online; accessed Apr 2023].
- NVIDIA. NVLink & NVSwitch: Fastest HPC Data Center Platform. <https://www.nvidia.com/en-us/data-center/nvlink/>, 2023d. [Online; accessed Apr 2023].
- Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
- Pjesivac-Grbovic, J. Towards automatic and adaptive optimizations of mpi collective operations. 2007.
- Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. Zero: memory optimizations toward training trillion parameter models. In Cuicchi, C., Qualters, I., and Kramer, W. T. (eds.), *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE/ACM, 2020.
- Rajbhandari, S., Li, C., Yao, Z., Zhang, M., Aminabadi, R. Y., Awan, A. A., Rasley, J., and He, Y. DeepSpeed-moe: Advancing mixture-of-experts inference and training to power next-generation AI scale. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2022.
- Riquelme, C., Puigcerver, J., Mustafa, B., Neumann, M., Jenatton, R., Pinto, A. S., Keysers, D., and Houlisby, N. Scaling vision with sparse mixture of experts. In *Proceedings of the Neural Information Processing Systems (NeurIPS)*, 2021.

- Roller, S., Sukhbaatar, S., Szlam, A., and Weston, J. Hash layers for large sparse models. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W. (eds.), *Proceedings of the Neural Information Processing Systems (NeurIPS)*, 2021.
- Sergeev, A. and Del Balso, M. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.
- Sharir, O., Peleg, B., and Shoham, Y. The cost of training NLP models: A concise overview. *CoRR*, abs/2004.08900, 2020.
- Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q. V., Hinton, G. E., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.
- Snir, M., Gropp, W., Otto, S., Huss-Lederman, S., Dongarra, J., and Walker, D. *MPI—the Complete Reference: the MPI core*, volume 1. MIT press, 1998.
- Thakur, R. and Choudhary, A. All-to-all communication on meshes with wormhole routing. In *Proceedings of 8th International Parallel Processing Symposium*, pp. 561–565. IEEE, 1994.
- Wikipedia. Ternary Search. [https://en.wikipedia.org/wiki/Ternary\\_search](https://en.wikipedia.org/wiki/Ternary_search), 2023. [Online; accessed Apr 2023].
- Yang, A., Lin, J., Men, R., Zhou, C., Jiang, L., Jia, X., Wang, A., Zhang, J., Wang, J., Li, Y., Zhang, D., Lin, W., Qu, L., Zhou, J., and Yang, H. Exploring sparse expert models and beyond. *CoRR*, abs/2105.15082, 2021.
- Zheng, L., Li, Z., Zhang, H., Zhuang, Y., Chen, Z., Huang, Y., Wang, Y., Xu, Y., Zhuo, D., Xing, E. P., Gonzalez, J. E., and Stoica, I. Alpa: Automating inter- and Intra-Operator parallelism for distributed deep learning. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2022.

## A TWO-DIMENSIONAL HIERARCHICAL (2DH) ALL-TO-ALL

This section describes 2DH All-to-All, a novel All-to-All algorithm proposed by TUTEL.

### A.1 Motivation: Small Size of Message Transfer

Most of popular DL frameworks (Microsoft, 2023; Ott et al., 2019; Sergeev & Del Balso, 2018; Paszke et al., 2019) leverage point-to-point (P2P) APIs of NCCL (NVIDIA, 2023b),<sup>4</sup> the state-of-the-art GPU collective communication library, to implement *Linear* All-to-All algorithm (see Algorithm 1). It operates on  $n$  GPUs, where each GPU splits its total  $S$  bytes of data into  $n$  chunks ( $S/n$  bytes each) and performs P2P communication with all other GPUs. The P2P chunk size  $S/n$  transferred between any two GPUs will become smaller when we scale out (larger  $n$ ), which is hard to saturate the high-speed links such as NVLink and HDR InfiniBand at a large scale (see Figure 16).  $S$  is fixed and only decided by the model itself.

### A.2 Approach and Challenges

To achieve a high link bandwidth, our approach is aggregating multiple data chunks that are sent from multiple local GPUs to the same remote GPU. This avoids sending multiple small messages over networking by merging small chunks into a single large chunk, which significantly improves the link bandwidth utilization.

Unfortunately, an efficient implementation of this approach on a large scale is challenging due to *the overhead of aggregating small messages*. Specifically, to aggregate chunks inside a node with  $m$  local GPUs, all  $m$  GPUs in the node need to exchange  $\frac{S}{n} \times \frac{n}{m} = \frac{S}{m}$  chunks with each other. This is equivalent to performing  $\frac{S}{n}$  size intra-node All-to-All  $\frac{n}{m}$  times, as illustrated in Figure 17, phase 1 of the naïve local aggregation All-to-All. The latency of this intra-node All-to-All process is expected to be constant as chunk size  $\frac{S}{m}$  does not rely on  $n$ , but unexpectedly, it actually increases as  $n$  scales out due to  $\frac{n}{m}$  times non-contiguous memory access on GPUs. For example, in phase 1 of the naïve local aggregation, intra-node GPUs exchange non-contiguous chunks twice with each other (01 and 05, 02 and 06, etc.) that incurs  $\mathcal{O}(\frac{n}{m})$  non-contiguous memory access on each GPU. Specifically, when  $S = 128$  MiB and  $m = 8$ , we observe that intra-node All-to-All process takes  $\sim 600\mu s$  for  $n = 8$  and increases up to  $\sim 5ms$  for  $n = 2048$ .

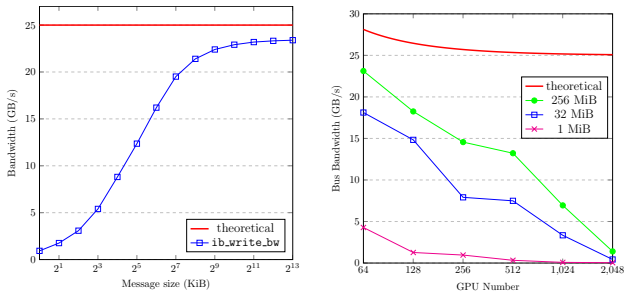
<sup>4</sup>Message Passing Interface (MPI) (Snir et al., 1998) also has developed various All-to-All algorithms (Pjesivac-Grbovic, 2007; Thakur & Choudhary, 1994; Bruck et al., 1997), but we only discuss NCCL in this work as it outperforms MPI in most DL scenarios. Note MPI mainly focuses on traditional HPC workloads where  $S$  is typically much smaller than DL workloads.

### Algorithm 1 Linear All-to-All using Point-to-Point APIs

```

1: procedure ALL2ALL_LINEAR(output, input)
2:    $n \leftarrow \text{ngpus}$ ,  $S \leftarrow \text{sizeof input}$ 
3:    $\text{chunksize} \leftarrow S / n$ 
4:   for  $r = 0$ ;  $r < n$ ;  $r++$  do
5:      $\text{loc} \leftarrow r \times \text{chunksize}$ ,  $\text{peer} \leftarrow r$ 
6:      $\text{ncclSend}(\text{input}[\text{loc}], \text{chunksize}, \text{peer})$ 
7:      $\text{ncclRecv}(\text{output}[\text{loc}], \text{chunksize}, \text{peer})$ 
8:   end for
9: end procedure

```



(a) GPUDirect RDMA `ib_write_bw` (TX depth = 8) over HDR InfiniBand on two Azure NDv4 VMs (Azure, 2023). (b) All-to-All bus bandwidth in `nccl-tests` scaling from 64-GPU to 2048-GPU.

Figure 16. Under-utilized bandwidth for small messages.

### A.3 Algorithm

To avoid the slowdown due to non-contiguous memory access, 2DH All-to-All consists of additional phases that conduct efficient stride memory copies to align non-contiguous chunks into a contiguous address space. To be specific, Figure 17 illustrates all phases of 2DH All-to-All in order. Instead of performing intra-node All-to-All from the beginning like the naïve local aggregation, we first align chunks that share the same local destination GPU via stride memory copies (phase 1) and then conduct intra-node All-to-All (phase 2). In the following phase, again, we align chunks that share the same remote destination GPU (phase 3) and then finally conduct inter-node All-to-All (phase 4). By leveraging stride memory copies, 2DH All-to-All achieves a high memory bandwidth utilization, keeping a constant and low latency regardless of  $n$  in the first three phases. The benefit of 2DH All-to-All over existing algorithms increases as  $S/n$  gets smaller (a smaller data size  $S$  or a larger number of GPUs  $n$ ). Note that this is beneficial for rail-optimized InfiniBand networking as well since it avoids cross-rail communication.

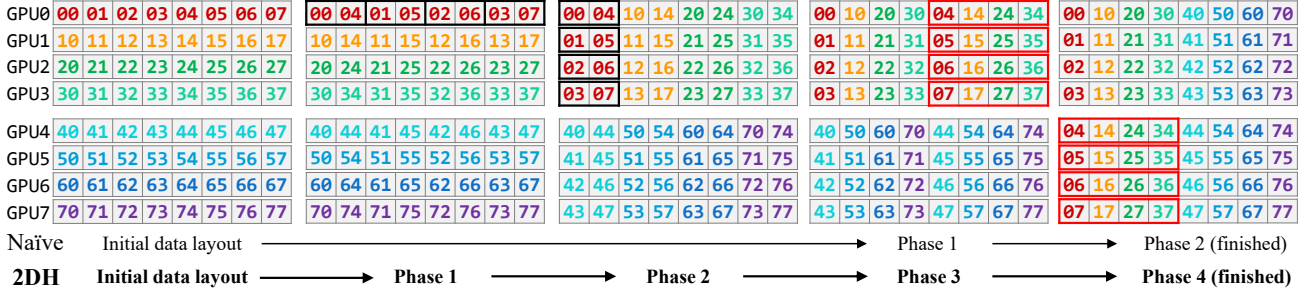


Figure 17. Example of data layouts in each phase of the naïve local aggregation All-to-All and two-dimensional hierarchical (2DH) All-to-All. In this example, there are two nodes that consist of GPU 0~3 and GPU 4~7, respectively.

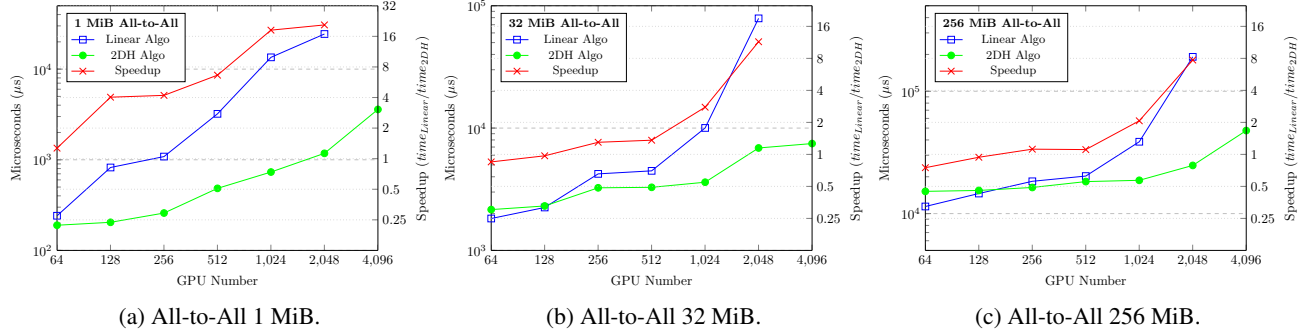


Figure 18. Comparison between linear and 2DH All-to-All algorithms with various sizes in NCCL.

#### A.4 Optimization with MSCCL

**Implementation using NCCL APIs.** We implement 2DH All-to-All algorithm using NCCL’s `ncclSend` and `ncclRecv` APIs (see details in Algorithm 2). It consists of two steps. The first step corresponds to phase 1 ~ 3 in Figure 17 and contains intra-node All-to-All communication and two stride memory copies, of which latencies only rely on  $S$ . The second step corresponds to phase 4 in Figure 17, which is inter-node All-to-All and its latency relies on  $n/m$  instead of  $n$  as local chunks are already merged.

**Optimization via MSCCL.** Implementation using NCCL APIs requires extra synchronization barriers between different phases in 2DH All-to-All and may cause throughput degradation. In order to achieve better performance, we leverage MSCCL by describing the 2DH algorithm in a domain specific language (DSL) and optimizing with the compiler (Cowan et al., 2023). The custom compiler also leverages LL128 protocol (NVIDIA, 2020a) for All-to-All, which could achieve better efficiency than default NCCL-based implementation in low latency scenarios like small sizes All-to-All.

**Extension.** On existing GPU clusters, local GPU number  $m$  is usually 8 or 16, which makes  $\frac{n}{m}$  still large when scaling out All-to-All to hundreds of thousands (100 K) of GPUs *at exascale*. The next generation

NVSwitch (NVIDIA, 2023d) enables up to 256 GPUs connected via high speed NVLink and makes it possible for 2DH All-to-All scaling out with  $m = 256$ . For large-scale network topologies like dragonfly (Kim et al., 2008), 2DH All-to-All could be further adapted to 3D by splitting inter-node to intra-group and inter-group All-to-All according to the network hierarchy.

#### A.5 Evaluation

We benchmark `alltoall_perf` in `nccl-tests` (NVIDIA, 2023c) to measure the performance and correctness of All-to-All operations. Experiment setup is as described in Section 5. The sizes of All-to-All start from 1 KiB and end at 16 GiB, with multiplication factor 2. The tests are launched via OpenMPI with proper NUMA binding. All of the All-to-All operations are out-of-place and correctness is also checked by `nccl-tests`. We compare the latency of specific sizes we are interested in between different algorithms and different implementations.

To illustrate scalability of the proposed 2DH All-to-All algorithm, we compare it with the state-of-the-art NCCL All-to-All in the same cluster. `alltoall_perf` in `nccl-tests` (NVIDIA, 2023c) uses the linear All-to-All algorithm by default while we also implement the 2DH All-to-All algorithm in `nccl-tests` to replace the original one. We scale the experiments from 64-GPU to 4096-GPU. As shown

**Algorithm 2** Two-Dimensional Hierarchical (2DH) All-to-All

```

1: procedure STRIDEMEMCPY(output, input, chunksize, row, col)
2:   for i = 0; i < row × col; i++ do
3:     j ← i % row × col + i / col
4:     output[j × chunksize : (j+1) × chunksize] ← input[i × chunksize : (i+1)
       × chunksize]
5:   end for
6: end procedure
7: procedure ALL2ALL_2DH(output, input)
8:   // step 1: intra-node All-to-All
9:   strideMemcpy(buffer, input, chunksize, ngpus_per_node, nnodes)
10:  for g = 0; g < ngpus_per_node; g++ do
11:    loc ← g × nnodes × chunksize, peer ← g + node_rank × ngpus_per_node
12:    ncclSend(buffer[loc], nnodes × chunksize, datatype, peer, comm)
13:    ncclRecv(output[loc], nnodes × chunksize, datatype, peer, comm)
14:  end for
15:  strideMemcpy(buffer, output, chunksize, nnodes, ngpus_per_node)
16:  // step 2: inter-node All-to-All
17:  for n = 0; n < nnodes; n++ do
18:    loc ← n × ngpus_per_node × chunksize, peer ← local_rank + n ×
       ngpus_per_node
19:    ncclSend(buffer[loc], ngpus_per_node × chunksize, datatype, peer, comm)
20:    ncclRecv(output[loc], ngpus_per_node × chunksize, datatype, peer, comm)
21:  end for
22: end procedure

```

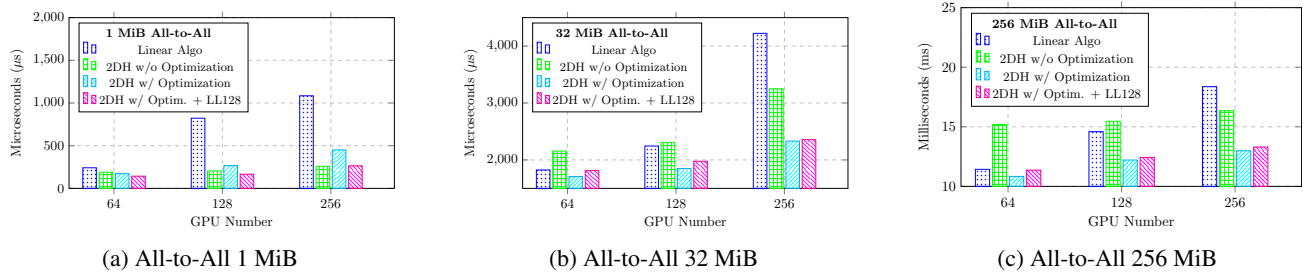


Figure 19. Comparison between NCCL and optimized implementation (Cowan et al., 2023) running 2DH All-to-All algorithm.

in Figure 18, the proposed 2DH algorithm could scale better with lower gradient than original linear algorithm. For small sizes (1 MiB), 2DH algorithm can achieve lower latency starting from small scales. For larger sizes (32 MiB and 256 MiB), 2DH algorithm has higher latency caused by extra data copies. While as the GPU number scales out, 2DH algorithm could perform better. Therefore, dynamic adaption between linear and 2DH algorithms is required. Besides, the 2DH algorithm can scale to 4096-GPU in our experiments while we didn’t run NCCL’s linear algorithm successfully in such large scale.

We also study the performance gain using the custom compiler (Cowan et al., 2023). As illustrated in Figure 19, the optimized implementation achieves better results than implementation using NCCL’s APIs. For example, 256 MiB

size on 64-GPU, 2DH algorithm in NCCL implementation has higher latency, but with the optimized implementation it could still outperform linear algorithm in NCCL. Besides, LL128 protocol has lower latency for small sizes (1 MiB and 32 MiB) while default protocol performs better for large sizes (256 MiB). Therefore, dynamic adaption between different protocols is necessary with this optimization.

## B SIMT-EFFICIENT FAST ENCODE AND DECODE

TUTEL implements sophisticated optimizations for the *encode* (generating All-to-All inputs out of MoE layer inputs during MoE dispatch) and *decode* (generating MoE layer



```

1 # Tensor shapes: logits(T,E)
2 gate_probs = softmax(logits)
3 # Tensor shapes: gate_probs(T,E), idxs(T,), scores(T,)
4 idxs, scores = top_k(gate_probs)
5 # Tensor shapes: locations(T,)
6 locations = compute_location(idxs)
7 # Tensor shapes: locations(T,), locations1(T,Cg)
8 locations1 = one_hot(locations, num_classes=Cg)
9 # Tensor shapes: gate_probs(T,E), combine(T,E,Cg)
10 combine = einsum("TE,TC->TE", gate_probs, locations1)
11 # Tensor shapes: dispatch_input(E,Cg,M), moe_input(T,M)
12 dispatch_input = einsum(
13     "TE,TC->ECM", bool(combine), moe_input)

```

(a) Dense implementation.

```

1 # Tensor shapes: logits(T,E)
2 gate_probs = softmax(logits)
3 # Tensor shapes: gate_probs(T,E), idxs(T,), scores(T,)
4 idxs, scores = top_k(gate_probs)
5 # Tensor shapes: locations(T,)
6 locations = compute_location(idxs)
7 # Tensor shapes:
8 # dispatch_input(E,Cg,M), moe_input(T,M)
9 dispatch_input = zeros((E,Cg,M))
10 for t in [0, 1, ..., T-1]:
11     # Broadcast multiplication
12     dispatch_input[idxs[t]][locations[t]] = \
13         bool(scores[t]) * moe_input[t]

```

(b) Sparse implementation.

Figure 20. Comparison between dense and sparse implementations of generating All-to-All dispatch input (dispatch\_input) out of an MoE layer input (moe\_input) and a gate function output (logits).

outputs out of All-to-All outputs during MoE combine) stages of an MoE layer. Existing implementations of encode and decode need einsum operations with a large time complexity, as described by GShard (Lepikhin et al., 2021) and implemented in Fairseq (Ott et al., 2019). For instance, Figure 20a shows the most heavy-weighted part of the encode implementation (decode is similar as encode since it is a reverse operation of encode). We observe that this implementation is unnecessarily dense as it contains a lot of zero multiplications and additions. TUTEL addresses this by a sparse implementation as shown in Figure 20b. Given that  $T$  is the number of input tokens per expert, while the time complexity of the dense version is  $O(T \cdot E \cdot C_g \cdot D)$ , the one of the sparse version is only  $O(T \cdot k \cdot D)$ , where  $T \cdot k = E \cdot C_g$  in most cases. This indicates that the sparse version has only  $1/T$  of time complexity than the dense version.

Unfortunately, it is challenging to implement efficient GPU kernels for the sparse implementation. While the dense computation can be dramatically accelerated by matrix multiplication accelerators (e.g., Tensor Cores), the sparse computation cannot leverage those accelerators efficiently.<sup>5</sup>

<sup>5</sup>Even the sparsity support by the latest hardware (e.g., 3rd-generation Tensor Cores) cannot work efficiently as it only sup-

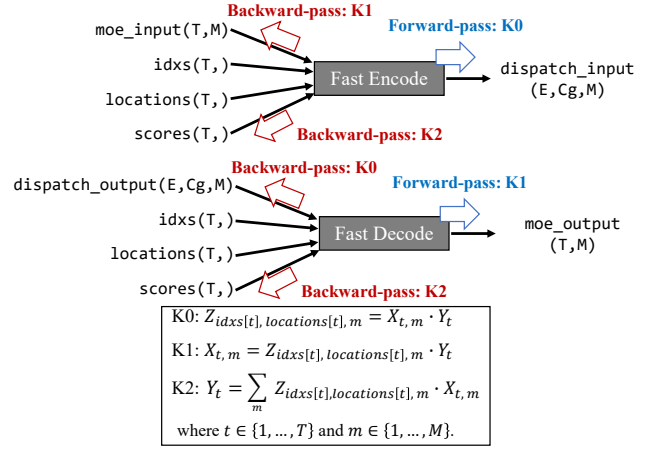


Figure 21. Forward- and backward-pass computations of fast encode and fast decode operators. Parentheses refer to tensor shapes. The tensor shapes of  $X$ ,  $Y$ , and  $Z$  are  $(T, D)$ ,  $(T,)$ , and  $(E, C_g, D)$ , respectively.  $idxs$  and  $locations$  have no backward-pass computation as they are not trainable inputs.

tokens/step	Fairseq MoE (GiB)	TUTEL MoE (GiB)
4,096	3.7	2.9 (-21.6%)
8,192	6.2	3.2 (-48.4%)
16,384	16.3	4.0 (-75.5%)
32,768	57.9	5.7 (-90.2%)

Table 9. GPU memory cost for single MoE layer. (Static Settings:  $D = H = 4096$ , top-k = 2,  $E_g = 2$ )

To tackle this issue, we implement differentiable fast encode and decode operators based on three specially designed GPU kernels: K0, K1, and K2, as illustrated in Figure 21. TUTEL accelerates these kernels by always assigning different indices of dimension  $T$  to different thread arrays (or warps), which ensures computation for a single token along dimension  $M$  is SIMT-efficient. By this approach, our sparse computation can actually leverage various optimizations that are applicable only for dense computation, such as warp shuffling, Blleloch scan algorithm, and element vectorization for low-precision computation (e.g., leveraging half2 types for half-precision computation). Aggregating all the kernel optimizations, TUTEL extremely minimizes the latency of encode and decode as shown in Figure 15. It greatly saves GPU memory as well. As shown in Table 9, in most cases, it achieves 20% ~ 90% memory saving. TUTEL exposes two interfaces for these optimized computations: `moe.fast_encode` used by MoE dispatch and `moe.fast_decode` used by MoE combine.

ports fine-grained sparsity, while our sparse computation belongs to coarse-grained sparsity (NVIDIA, 2020b).

Method	$E$	$k$	$f$	MoE	$AP^{box}$	$AP^{mask}$
SwinV2-B	-	-	-	-	53.0	45.8
SwinV2-MoE-B	32	1	1.25	tuned	51.3 (-1.7)	44.4 (-1.4)
SwinV2-MoE-B	32	1	1.25	fixed	53.4 (+0.4)	46.2 (+0.4)

Table 10. The results on COCO object detection. “fixed” MoE indicates that the MoE layers are fixed in fine-tuning.

## C MORE RESULTS ON SWINV2-MOE

### C.1 How to do fine-tuning on COCO object detection?

Previous MoE models on computer vision only perform experiments using image classification tasks (Riquelme et al., 2021). It is unclear whether the sparse MoE models perform well on down-stream computer vision tasks as well such as COCO object detection.

As shown in Table 10, direct fine-tuning will result in poor performance, with -1.7/-1.4 box/mask AP drops compared to the dense counterparts. We find that fixing all MoE layers in fine-tuning can alleviate the degradation problem, and we obtain +0.4/+0.4 box/mask AP improvements by this strategy.

Also note it is the first time that a sparse MoE model is applicable and superior on the important computer vision tasks of COCO object detection. We hope TUTEL to empower more down-stream AI tasks.

### C.2 Ablation Study

**Ablation on Number of Experts.** Table 11 ablates the effect of expert number, using different model sizes (SwinV2-S and SwinV2-B) and a variety of vision tasks. It can be seen that 32 and 64 perform the best, which is consistent with that in previous works (Riquelme et al., 2021; Du et al., 2022).

**Comparison of Routing Algorithms and Capacity Factors.** Figure 22 compares the routing methods with and without batch prioritized routing (BPR) (Riquelme et al., 2021). It shows that the BPR approach is crucial for computer vision MoE models, especially at lower capacity factor values. These results are consistent with reported in (Riquelme et al., 2021).

Table 12 ablates the performance of SwinV2-MoE model given different  $k$  and capacity factor  $f$ . It is observed that top-1 router has a better speed-accuracy trade-off. We use default hyper-parameters of  $k = 1$  and  $f = 1.0$ .

### C.3 A New Cosine Router Supported in TUTEL

With TUTEL, we provide more MoE baselines to enrich the algorithm choices and to exemplify how to leverage this

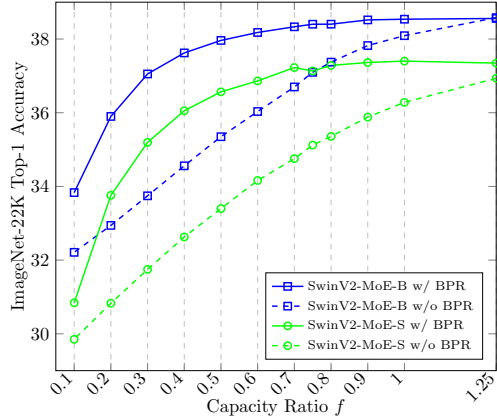


Figure 22. ImageNet-22K top-1 accuracy w.r.t inference capacity factor. “w/ BPR” indicates training with batch prioritized routing while “w/o BPR” not. All models are trained on the ImageNet-22K dataset with  $E = 32$ ,  $k = 1$ ,  $f = 1.25$  and an input resolution of  $192 \times 192$  for 90 epochs.

framework for algorithmic innovation. One attempt is a new cosine router that hopes to improve numerical stability with increased model size, inspired by (Liu et al., 2022):

$$P = \text{Softmax}\left(\frac{W\mathbf{x} \cdot M}{\|W\mathbf{x}\| \|M\|} / \tau\right), \quad (2)$$

where  $W \in \mathbb{R}^{D \times C}$  is a linear layer used to project the input token feature  $x \in \mathbb{R}^{C \times 1}$  to dimension  $D$  (256 by default);  $M \in \mathbb{R}^{E \times D}$  is a parametric matrix, with each column representing each expert;  $\tau$  is a learnable temperature that is set lowest 0.01 to avoid temperatures being too small;  $P$  denotes the routing scores for selecting experts.

Our preliminary experiments in Table 13 show that when using 32 experts, the cosine router is as accurate in image classification as a common linear router. Although it is not superior in image classification at the moment, we still encourage TUTEL users to try this option in their problems, because: 1) its normalization effect on input may lead to more stable routing when the amplitude or dimension of the input feature is scaled; 2) There is a concurrent work showing that the cosine router is more accurate in cross-lingual language tasks (Chi et al., 2022).

TUTEL: Adaptive Mixture-of-Experts at Scale

Method	$E$	$k$	$f$	#param	#param <sub>act</sub>	GFLOPs	Train speed	Inference speed	IN-22K acc@1	IN-22K train loss	IN-1K/ft acc@1	IN-1K/5-shot acc@1
SwinV2-S	-	-	-	65.8M	65.8M	6.76	350	1604	35.5	5.017	83.5	70.3
SwinV2-MoE-S	8	1	1.0	173.3M	65.8M	6.76	292	1150	36.8 (+1.3)	4.862	84.5 (+1.0)	75.2 (+4.9)
SwinV2-MoE-S	16	1	1.0	296.1M	65.8M	6.76	295	1153	37.5 (+2.0)	4.749	84.9 (+1.4)	76.5 (+6.2)
SwinV2-MoE-S	32	1	1.0	541.8M	65.8M	6.76	295	1159	37.4 (+1.9)	4.721	84.7 (+1.2)	75.9 (+5.6)
SwinV2-MoE-S	64	1	1.0	1033M	65.8M	6.76	288	1083	37.8 (+2.3)	4.669	84.7 (+1.2)	75.7 (+5.4)
SwinV2-MoE-S	128	1	1.0	2016M	65.8M	6.76	273	1027	37.4 (+1.9)	4.744	84.5 (+1.0)	75.4 (+5.1)
SwinV2-B	-	-	-	109.3M	109.3M	11.78	288	1195	37.2	4.771	85.1	75.9
SwinV2-MoE-B	8	1	1.0	300.3M	109.3M	11.78	247	893	38.1 (+0.9)	4.690	85.3 (+0.2)	77.2 (+1.3)
SwinV2-MoE-B	16	1	1.0	518.7M	109.3M	11.78	246	889	38.6 (+1.4)	4.596	85.5 (+0.4)	78.2 (+2.3)
SwinV2-MoE-B	32	1	1.0	955.3M	109.3M	11.78	249	892	38.5 (+1.3)	4.568	85.5 (+0.4)	77.9 (+2.0)
SwinV2-MoE-B	32	2	1.0	955.3M	136.6M	11.78	206	679	38.6 (+1.4)	4.506	85.5 (+0.4)	78.7 (+2.8)
SwinV2-MoE-B	32	2	0.625	955.3M	136.6M	12.54	227	785	38.3 (+1.1)	4.621	85.2 (+0.1)	77.5 (+1.6)

Table 11. Comparison of SwinV2-MoE models and the dense counterparts (Liu et al., 2022). The sparse MoE model is obtained by replacing the FFN of every other layer with an MoE layer.  $E$  denotes the number of experts in the MoE layer.  $k$  denotes the number of selected experts per token.  $f$  denotes the capacity factor. The “Train speed” and “Inference speed” are measured by images per second during training and inference. All models are trained on the ImageNet-22K dataset with an input resolution of  $192 \times 192$ . We report the top-1 accuracy and final training loss on ImageNet-22K classification (IN-22K), the fine-tuning top-1 accuracy on ImageNet-1K classification (IN-1K/ft) and the 5-shot linear evaluation top-1 accuracy on ImageNet-1K classification (IN-1K/5-shot). Also note that TUTEL supports multiple GPUs to share one expert, which empowers us to leverage 32 GPUs for the experiments with expert number as 8 and 16.

Method	$k$	Train- $f$	Infer- $f$	Infer GFLOPs	Infer speed	IN-22K acc@1
SwinV2-B	-	-	-	11.78	1195	37.2
SwinV2-MoE-B	1	1.0	1.25	12.54	839	38.6 (+1.4)
SwinV2-MoE-B	1	1.0	1.0	11.78	892	38.5 (+1.3)
SwinV2-MoE-B	1	1.0	0.625	10.65	976	38.2 (+1.0)
SwinV2-MoE-B	1	1.0	0.5	10.27	1001	38.0 (+0.8)
SwinV2-MoE-B	2	1.0	1.25	16.31	621	38.7 (+1.5)
SwinV2-MoE-B	2	1.0	1.0	14.80	679	38.6 (+1.4)
SwinV2-MoE-B	2	1.0	0.625	12.54	785	38.4 (+1.2)
SwinV2-MoE-B	2	1.0	0.5	11.78	826	38.3 (+1.1)
SwinV2-MoE-B	2	0.625	0.625	12.54	785	38.3 (+1.1)
SwinV2-MoE-B	2	0.625	0.5	11.78	826	38.3 (+1.1)

Table 12. Ablations of top- $k$  and capacity factors  $f$ . “Train- $f$ ” and “Infer- $f$ ” indicates the capacity factor during training and inference. “Infer GFLOPs” and “Infer speed” indicates the GFLOPs and real speed (images/second) during inference.

Method	Router	IN-22K acc@1	IN-1K/ft acc@1	IN-1K/5-shot acc@1
SwinV2-S	-	35.5	83.5	70.3
SwinV2-MoE-S	Linear	37.4 (+1.9)	84.7 (+1.2)	75.9 (+5.6)
SwinV2-MoE-S	Cosine	37.1 (+1.6)	84.3 (+0.8)	75.2 (+4.9)
SwinV2-B	-	37.2	85.1	75.9
SwinV2-MoE-B	Linear	38.5 (+1.3)	85.5 (+0.4)	77.9 (+2.0)
SwinV2-MoE-B	Cosine	38.5 (+1.3)	85.3 (+0.2)	77.3 (+1.4)

Table 13. Comparison between the linear router and cosine router ( $E = 32, k = 1, f = 1.25$ ).