背景：MoE 的动态特性和系统的静态并行/ pipeline 方式不匹配，限制了大规模 MoE model 的实现
TUTEL 是一个高可扩展的 MoE 全栈系统，实现了 MoE model 实时的自适应并行策略 (高效的 expert 执行，单个 expert <- 多 GPUs) 和自适应 pipeline 策略 (高效的 MoE dispatch 和 combine)；同时，TUTEL 还包括一个用来加速 MoE 通信的二维多级 all-to-all 算法，以及一个 flexible all-to-all 算法。

TUTEL: Adaptive Mixture-of-Experts at Scale

Changho Hwang[1,†], Wei Cui[1,†], Yifan Xiong[1,†], Ziyue Yang[1,†], Ze Liu[1], Han Hu[1], Zilong Wang[2], Rafael Salas[2], Jithin Jose[2], Prabhat Ram[2], Joe Chau[2], Peng Cheng[1], Fan Yang[1], Mao Yang[1], and Yongqiang Xiong[1]

[1]Microsoft Research [2]Microsoft

Abstract

In recent years, Mixture-of-Experts (MoE) has emerged as a promising technique for deep learning that can scale the model capacity to trillion-plus parameters while reducing the computing cost via sparse computation. While MoE opens a new frontier of exceedingly large models, its implementation over thousands of GPUs has been limited due to mismatch between the dynamic nature of MoE and static parallelism/pipelining of the system.

We present TUTEL, a highly scalable stack design and implementation for MoE with dynamically adaptive parallelism and pipelining. TUTEL delivers adaptive parallelism switching and adaptive pipelining at runtime, which achieves up to 1.74× and 2.00× single MoE layer speedup, respectively. We also propose a novel two-dimensional hierarchical algorithm for MoE communication speedup that outperforms the previous state-of-the-art up to 20.7× over 2,048 GPUs. Aggregating all techniques, TUTEL finally delivers **4.96×** and **5.75×** speedup of a single MoE layer on 16 GPUs and 2,048 GPUs, respectively, over Fairseq: Meta's Facebook AI Research Sequence-to-Sequence Toolkit (TUTEL is now partially adopted by Fairseq). TUTEL source code is available in public: https://github.com/microsoft/tutel.

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. SwinV2-MoE is open sourced in https://github.com/microsoft/Swin-Transformer.

Expert Capacity 被定义为 expert 上的 workload

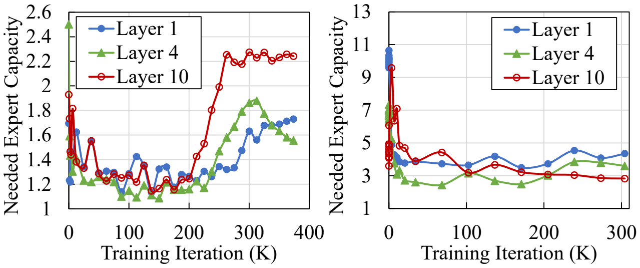


Figure 1: Dynamically changing workload of MoE layers during a full end-to-end training of the MoE version of Swin Transformer V2 [22, 23] thin-tiny (left) and base (right) models. The y-axis is the needed expert capacity at runtime, which is equivalent to the normalized amount of workload (see details 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.

1 Introduction

From the fast growth of machine learning (ML) techniques driven by deep neural networks (DNNs) in the past few years, the community finds that enrolling more DNN model parameters is one of the most straight-forward but less sophisticated way to improve the performance of ML algorithms [15]. However, DNN model capacity is often limited by computing resource and energy cost [39]. The root cause is the *dense* architecture of DNNs where the computing cost typically scales linearly to the number of parameters.

To tackle this, Mixture-of-Experts (MoE) [14] is adopted into DNNs that 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 (hence higher model accuracy) at only a little additional cost as it enrolls more model parameters (i.e., experts) without extra computation. Nowadays, MoE is one of the most popular approaches demonstrated to scale DNNs to trillion-plus parameters [9, 10, 18, 20, 36, 40], paving the way for models capable of learning even more information.

[†] Equal contribution.

While MoE-based algorithms open up a huge scale-up/out opportunity, it introduces fundamental system-side challenges that have not been seen before in most of previous deep learning (DL) algorithms and systems. The root cause is the **dynamic nature of MoE**. 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 each expert is fundamentally uncertain – it depends on input data and the gating function. Both of them change at every iteration in practice. In our experiments, the workload changes up to 4.38× in a single training (see Figure 1) and is also quite diverse for different layers.

MoE 的动态特性：一个 MoE layer 由多个分布在不同 GPUs 上的并行 experts 组成，而每个 expert 的 workload 是无法确定的，依赖于输入和 gate 函数，进而不断改变。

上述动态特性导致 MoE experts 经常无法采用性能最优的并行和 pipeline 策略，因为最优策略一直在动态变化

Existing DL systems, including the latest MoE frameworks [12, 17, 18, 33], are mostly based on static runtime execution that does not fit dynamic MoE characteristics. The pitfall comes from that experts often fail to leverage the best-performing parallelism and pipelining strategies as the optimal one differs depending on the dynamic workload. It is non-trivial to dynamically adjust parallelism and pipelining at runtime as it typically incurs a large runtime overhead or GPU memory consumption in existing systems. Furthermore, the computation-side optimization highly depends on the communication-side optimization, which needs to be optimized at the same time.

动态实时调整并行和 pipeline 策略并不容易，会造成高延迟和 GPU 显存开销；同时，还需要对通信结构进行优化

This paper presents TUTEL, an MoE system to fully optimize MoE layer performance by adaptive methods for dynamic MoE workload at any scale. The mechanism consists of two key techniques: adaptive pipelining for efficient MoE dispatch/combine and adaptive parallelism switching for efficient expert execution. Besides, TUTEL introduce a novel two-dimensional hierarchical (2DH) All-to-All algorithm and flexible All-to-All to enable efficient MoE dispatch/combine in exa-scale (4,096 A100 GPUs). TUTEL is a fully implemented framework for diverse MoE algorithms at scale. It has been open sourced on GitHub https://github.com/microsoft/tutel and already been integrated into Fairseq [33] and DeepSpeed [1]. Our extensive experiments over Azure A100 clusters [4] show that with 128 GPUs, TUTEL delivers up to 3.11× of MoE-layer speedup, and 1.55×/ 2.11× speedup for end-to-end training / inference of a real-world model (SwinV2-MoE), compared to that of using the original Fairseq. For 2,048 GPUs, the MoE-layer speedup is further improved to 5.75×.

Our key contributions are as follows:

- Provide detailed analysis on the dynamic nature of MoE and the following challenges in existing ML frameworks.
- Propose adaptive parallelism switching and adaptive pipelining to handle dynamic workloads of MoE efficiently, which achieve up to 1.74× and 2.00× speedup on a single MoE layer, separately.
- Propose the novel 2DH All-to-All algorithm and flexible All-to-All, which outperform the previous state-of-the-art dispatch up to 20.7× on 2,048 A100 GPUs, and enables MoE on 4,096 A100 GPUs in exa-scale.
- 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× and 2.11× 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.

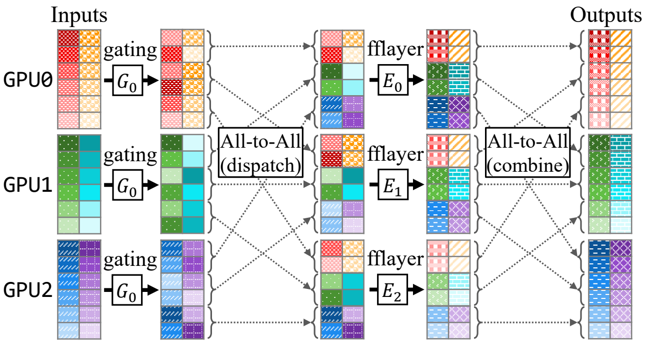


Figure 2: Example of an MoE layer across three GPUs, expert E_i on GPU i. G_0 represents the gating function that is shared across all GPUs. Different colors or patterns indicate different samples (columns of inputs) and different gradients of color indicate different tokens within a sample (rows of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

2 Background & Motivation

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

2.1 Background

Mixture-of-Experts for Deep Learning. Mixture-of-Experts (MoE) [14] is an ML concept that employs multiple *expert* models, which deal with their own specialized sub-tasks respectively to solve the entire tasks together. While MoE has been widely adopted by many classic ML algorithms [44], its concept is recently applied to large-scale distributed DNN models [10, 20, 36] by putting a cross-GPU layer that partially exchanges hidden features from different GPUs, which is often called an MoE layer. Figure 2 illustrates an example. First, it runs a *gating function* [19, 37, 43] that determines the destination GPU of each input token[2] in the following all-to-all collective communication (All-to-All).

[2] Each input sample should be divided into one or more tokens, and the definition of a token depends on the model's algorithm and tasks.

这里的将一个 sample 划分为多个 tokens 应该是适用于 NLP RNN 领域的，例如将一个长句子划分为多个短语，图 2 中 all-to-all 之后也不是将不同 samples 的不同 tokens 组合，只是利用 GPU 的多 sm 来分别并行计算而已

MoE 中的每个 sample 都要被分成一个或多个 tokens；

MoE Dispatch: 通过一个 all–to–all 的 collective 通信，决定每个输入 token 的目标 GPUs (一个或多个目标 GPUs，进行计算，每个 GPU 运行一个 expert)；

MoE Combine: 通过第二个 all–to–all 的 collective 通信，将每个 token 对应的输出发送回原来的 GPU；

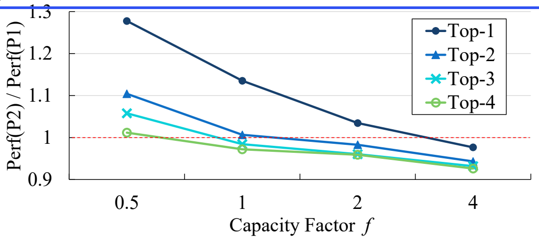


Figure 3: Runtime preferences of two different parallelism methods. The figure compares their throughput under varying capacity factor f (i.e., varying amount of workload) and different top-k configurations, where above 1.0 implies that P2 outperforms P1, and vice versa. The model uses 16K fflayer hidden size, 2,048 fflayer channel size, and 4 batch size.

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 [9, 10, 18, 36]. Currently, many state-of-the-art frameworks (e.g., DeepSpeed [1], Fairseq [33], etc.) have already supported MoE.

MoE 的高 cost –efficiency: 在超大规模 DL 中，增加模型参数(即增加 experts) 不会显著增加每个 token 的计算开销

Dynamic workload of MoE. The root cause of dynamic workload of MoE is due to two reasons. 1) **Token routing mechanism.** MoE layers route each token to multiple experts dynamically and 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. 2) **Dynamic expert capacity.** The workload of each expert is capped by *expert capacity*, which is defined as the maximum number of tokens for one expert. 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 capacity factor f ($f \geq 1$) as follows:

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

f is dynamically adjusted during training, which contributes to dynamic workload – it is increased/decreased when the token distribution is uneven/even [10, 19].

MoE 动态负载的原因：

1) Token routing 机制。动态，经常在 experts 间不均衡；

2) 动态 expert capacity。表征单个 expert 接收到的最大 tokens 数目，被定义为 k (from top–k) * f (容量因子) * T (每个 batch 内的 tokens 数目) / E (experts 数目)

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) [7, 10, 17]. 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.

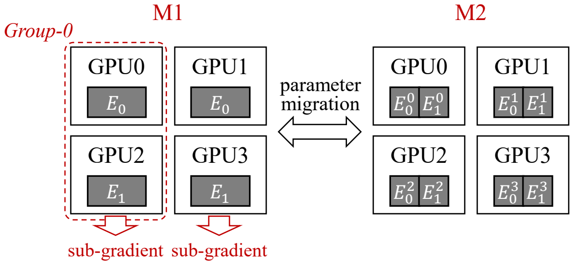


Figure 4: Simple parallelism methods for MoE. E_i^p refers to p-th partition (model-parallel manner) of i-th expert (no p means not partitioned). M1 replicates each expert on two GPUs each, and M2 partitions each expert across four GPUs respectively.

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%
Potential speedup	**1.51**×	**1.86**×	**1.76**×

Table 1: Ratio of All-to-All overhead and potential speedup by fully overlapping All-to-All and computation in a typical MoE setting.

增大 experts 数目会在单个 expert 上带来显著的边际递减效应，因此大规模训练中仅采用远少于 GPU 数目的 experts 数目，且单个 expert <– 多个 GPUs

According to our experiments, static parallelism methods do not always work efficiently under dynamic workload. We run a single MoE layer using optimized parallelism P1 and P2 (details in Section 3.2). As shown in the Figure 3, the best parallelism method depends on the dynamic workload, which has 7.39%-27.76% performance gap between these two parallelisms. **Therefore, an adaptive parallelism is beneficial for efficient MoE training.**

自适应并行对提高不同配置下的 MoE 训练性能 (吞吐) 很有帮助，但并行策略间的切换开销很大

Unfortunately, adaptive parallelism switching is difficult in existing systems. Different parallelism strategies such as M1 or M2 shown in Figure 4 require storing different sets of model parameters in each GPU. Therefore, switching parallelism at runtime would incur a large runtime overhead for parameter migration. Furthermore, different parallelism strategies may have their own restrictions on accessing inputs, synchronizing expert gradients, etc., which also complicates switching them with other strategies.

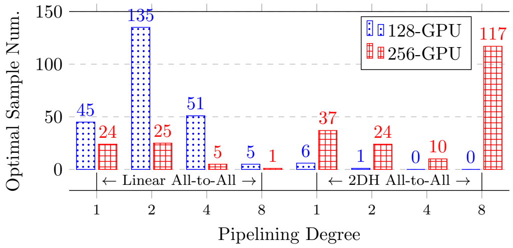


Figure 5: The distribution of optimal pipeline strategies for various MoE workload samples. Each column indicates the number of samples that perform best with the strategy described on X-axis. Details of workload samples are in Table 6. Linear/2DH All-to-All refers to two different All-to-All algorithms we implement, which are explained in Section 3.4.

2.3 Static Pipelining

静态 pipeline 使用静态 all-to-all 算法和固定的 pipeline degree，将 all-to-all (主要是跨 GPU 的内存拷贝，计算不密集) 和 expert (fflayer) 计算进行 pipeline (overlapping)，以提高利用率

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. Specifically, Table 1 shows up to 1.86× potential speedup by overlapping All-to-All and fflayer computation.

然而，在不同 MoE 配置和规模下，静态策略不总能获得最优性能

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 and pipelining degrees. This 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 the varying settings.

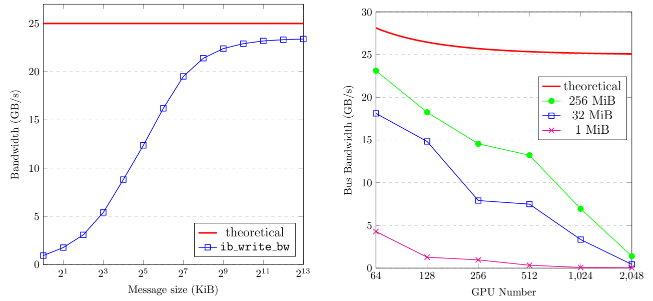
此外，GPU 上的计算和通信会产生相互干扰，影响获得最优 pipeline 策略，这是因为在相同 GPU 上同时运行 NCCL kernels 和 computation kernels 的干扰难以评估。实验表明，即使两个不同的 all-to-all 算法单独运行时吞吐相同，在引入相同 computation kernel 后吞吐也会显著不同

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.

2.4 Non-scalable MoE Dispatch & Combine

While All-to-All is the key collective communication primitive in MoE for dispatch and combine, we observe that existing All-to-All implementations perform poorly at a large scale. We elaborate in the following paragraphs.

现有 all-to-all 实现在大规模场景下表现很差。



(a) GPUDirect RDMA `ib_write_bw` (TX depth = 8) over HDR InfiniBand on two Azure NDv4 VMs [4].

(b) All-to-All bus bandwidth in nccl-tests scaling from 64-GPU to 2048-GPU.

Figure 6: Under-utilized bandwidth for small messages.

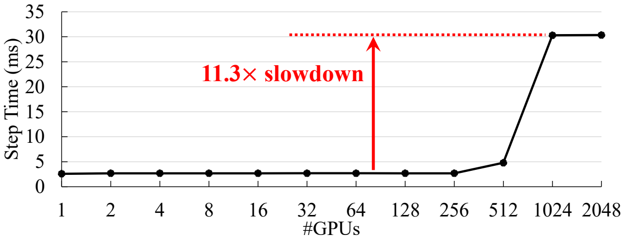


Figure 7: DeepSpeed fflayer elapsed time in different scales. $\Delta E = 1$, $M = V = 2048$, $f = 1$, tokens/step = 16384, W = #GPUs, op = *bgemm_strided_batched*.

All-to-all 过程一般利用 NCCL 支持的 GPUs 间 P2P 通信 APIs，通信量为 model size / GPU nums。因此，当 scale out 扩大 GPU 数目时，通信量会显著下降，进而造成 high-speed links (e.g., NVLink, HDR InfiniBand) 的利用率下降

Small Size of Message Transfer. Most of popular DL frameworks [1, 33, 34, 38] leverage point-to-point (P2P) APIs of NCCL [31],[3] 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 6). S is fixed and only decided by the model itself.

所以，在 MoE 模型面向的大规模 (GPU 数目多) 和小 workload (model 计算和通信稀疏) 场景中，all-to-all 通信很难适用

Unlike state-of-the-art all-reduce implementations that select different communication algorithms depending on the data size and the networking topology [27], All-to-All in existing DL systems has been using only a single algorithm that is descent at a large workload and a small scale but performs poorly at a small workload and a large scale. This makes the communication of MoE difficult to adapt to the dynamic workload, especially at a large-scale where MoE-based models typically target for.

[3] Message Passing Interface (MPI) [41] also has developed various All-to-All algorithms [5, 35, 42], 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 ← ngpus, S ← sizeof input
3:     chunksize ← S / n
4:     for r = 0; r < n; r++ do                          ▷ ncclGroupStart
5:         loc ← r × chunksize, peer ← r
6:         ncclSend(input[loc], chunksize, peer)
7:         ncclRecv(output[loc], chunksize, peer)
8:     end for                                           ▷ ncclGroupEnd
9: end procedure
```

Symbol	Description
W	the world size used for All-to-All exchange
M	fflayer channel size for each sample
V	fflayer hidden size for each sample
ΔE	the number of local experts per GPU
E	the number of global experts
ΔC	per-GPU tokens within local capacity limit
C	the gather of every ΔC
f	the capacity factor used in Equation (1)

Table 2: Symbol description in MoE dispatch and combine.

Inappropriate Computation Layout. We observe that the scalability issue in existing systems is not only about the communication but also comes from the computation. For example, in Figure 7, we measure the pure fflayer computation time on DeepSpeed MoE layer with varying numbers of GPUs. The computation time increases to 30.2 ms on 2,048 GPUs, which is 11.3× slowdown compared with the one of a single GPU. After profiling, we find that the performance regression is due to the rigid output layout produced by All-to-All primitive. Specifically, in Figure 7, when the number of GPUs grows from 1 to 2,048, the matrix multiplication conducted by fflayer changes from $A(1,\Delta E,16384,M)\cdot W(\Delta E,M,V)$ to $B(2048,\Delta E,8,M)\cdot W(\Delta E,M,V)$ (parentheses indicate a tensor shape, see symbol descriptions in Table 2). Note the huge difference of the 3rd dimension of input tensors, which has a large impact on the efficiency of matrix multiplication on GPU – e.g., PyTorch computes these using *bgemm_strided_batched* operation, where the later one achieves only 8.8% of computational throughput compared with the former one. This finding implies that we need to care about the tensor layout transformation incurred by All-to-All to achieve high scalablity of MoE layers.

当 scale out 获得更大 GPU 数目时，expert fflayer 计算用的 tensor shape 也会变小 (batch 维度变大，每个 sub-sample 变小)。虽然每个 GPU 执行的 sub-sample 变小，但 GPUs 需要共享部分计算结果 (对于矩阵乘操作)，计算并不完全并行，因此整体吞吐量还是会在 GPU 数目变得极端大时下降

上述分析应该不对，不然不应该 GPU num 刚开始增加的时候 step time 不变，应该就是 Pytorch batched 矩阵乘函数的底层实现导致的

3 TUTEL System Design

TUTEL, a full-stack MoE system, supports a complete MoE layer with adaptive optimizations. Because 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. Figure 8 shows a sample code using TUTEL's Python interface.

```python
from tutel import moe
from tutel import net

def custom_moe(x, top_k=2):
    scores = softmax(CustomGate(x), dim=1)
    crit, l_aux = moe.top_k_routing(scores, top_k)
    y = moe.fast_encode(x, crit)
    y = net.flex_all2all(y, 1, 0)
    y = CustomExpert(y)
    y = net.flex_all2all(y, 0, 1)
    output = moe.fast_decode(y, crit)
    return output, l_aux
```

Figure 8: Example of a custom MoE layer implemented using the TUTEL user interface.

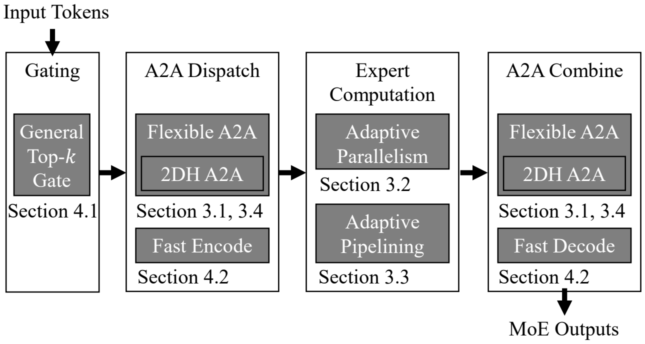


Figure 9: TUTEL contributions overview. A2A is an abbreviation of All-to-All.

In the following subsections, we describe how TUTEL tackles the aforementioned problems in detail. Figure 9 illustrates the major optimizations of TUTEL in each part of an MoE layer. In Section 3.1, we design *Flexible All-to-All* to ensure no computation regression in difference scales and guarantee the robustness for adaptive optimizations. In Section 3.2 and Section 3.3, we introduce *Adaptive Parallelism Switching* and *Adaptive Pipelining* to optimize end-to-end MoE performance on dynamic workload at any scale. Finally, in Section 3.4, we propose *Two-Dimensional Hierarchical All-to-All* to enable MoE training in exa-scale.

Operation	Transform Layout
$all2all(input)$	$(E, \Delta C, M) \rightarrow (W, \Delta E, \Delta C, M)$
$flex\_all2all(input, 1, 0)$	$(E, \Delta C, M) \rightarrow (\Delta E, C, M)$
$flex\_all2all(input, 0, 1)$	$(\Delta E, C, M) \rightarrow (E, \Delta C, M)$

Table 3: Output Layer by All-to-All and Flexible All-to-All

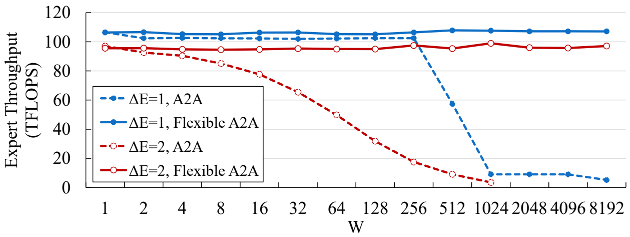


Figure 10: Throughput for expert computation based on A2A (All-to-All) layout and Flexible A2A layout.

3.1 Flexible All-to-All

The data exchange performed by All-to-All can be expressed as a transformation between tensor layouts – used by MoE dispatch, All-to-All primitive transforms its input tensor layout from $(E, \Delta C, M)$ to $(W, \Delta E, \Delta C, M)$ whose shape differs at different scales. While most existing frameworks (e.g., Deepspeed and Fairseq) directly use this output for following computation (e.g., custom expert fflayer), this is one potential reason that slows down MoE throughput at large scale due to inefficient expert computation at certain shapes.

TUTEL tackles this efficiency regression by using a different abstraction of All-to-All interface which is called *Flexible All-to-All*. According to Table 3, *Flexible All-to-All* extends two more arguments based on the original All-to-All. Apart from its first argument as input data to transform, the second argument specifies the tensor dimension to concatenate, and the third argument specifies the tensor dimension to split. This abstraction is similar to All-to-All from JAX [11]. In the MoE dispatch, the special role played by *Flexible All-to-All* is to keep output layout $(\Delta E, C, M)$ not relying on W. This ensures that the following computations after *Flexible All-to-All* can keep an unchanged layout for different scales. Moreover, to avoid computation regression caused by dynamic capacity changes, *Flexible All-to-All* can further tile the workload and bind computation into a well-tuned size if necessary, e.g., $(\Delta E, T, M)$, where T is the constant tile size used to split the workload into. Figure 10 shows the changes of expert computation time before and after fixing the layout.

将 all–to–all 传完的 tensor shape 中的 batch 维度（表征 GPU 数目）给融合到 Experts num 维度或 tokens num 维度，均为 per–GPU num –> global num。在 Pytorch batched 矩阵乘函数的底层实现（具体是为啥有待考证）中，这样得到的 tensor shape 会带来更大的吞吐（与 GPUs 间需要共享部分计算结果导致的延迟无关）

3.2 Adaptive Parallelism Switching

A static parallelism strategy is inefficient for MoE where the workload of experts dynamically changes. We expect multiple parallelism to be switchable at runtime to satisfy optimal strategy in different MoE training iterations, which is so called switchable parallelism.

GPU 0 和 1 是一组（使用 ZeRO 存储部分不同参数），GPU 2 和 3 是一组

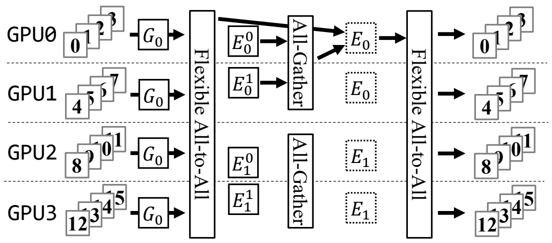


Figure 11: Switchable Expert + Data Parallelism.

TUTEL specially designs a pair of hybrid parallel strategies leveraging advantages from both data and model parallelism, and the most important point is that they are dynamically switchable at zero cost. Different parallel strategies are designed to have the same preference in token feeding, gradient updating, and parameter placement, allowing them to switch into each other instantly according to the dynamic change of *top-k* and *capacity-factor*.

不同并行策略（DP, MP）间的切换无额外开销，不同并行策略对 token feeding，梯度更新和参数放置均有相同偏好，因此可以根据 top–k 和容量因子而动态改变

Switchable Expert + Data Parallelism (P1). As shown in Figure 4, trivial expert + data parallelism divides GPUs into multiple groups: a. within each group, it works as a standard expert parallelism; b. different groups replicate expert copies to work in data parallelism, expecting each replicated group to produce a sub-gradient for the replicated expert.

TUTEL redesigns this parallelism not only for better coverage but also for switchablity. According to Figure 11, All-to-All within each data replicas group are fused as a single global All-to-All. Meanwhile, each GPU only maintains a slice of expert parameter in ZeRO style, whose replicas format can be temporarily accessed by all-gather communication. This design guarantees that expert parameters on each GPU is globally unique, while the theoretical communication size still keeps equivalent with trivial expert + data parallelism, which is $T_{data} = O(\Delta E \cdot C \cdot M) + O(parameters\_in\_single\_expert)$.

Trivial DP：将 GPUs 分为多个组，每个组内部搞 expert parallelism，组间搞 DP，all–to–all 的作用范围为组内；

TUTEL DP：允许跨组的 flexible all–to–all，这样有更好的收敛性能，且通信量不变

Switchable Expert + Model Parallelism (P2). Switchable Expert + Model Parallelism is another method align with P1 at runtime. It consumes less communication on expert parameters but more communication on input tokens. Thus, P2 can perform faster than P1 when the token size is relatively smaller than expert parameter size.

According to Figure 12, MoE dispatch requires a local repeat operation before All-to-All to copy tokens *n-sharded* times, where *n-sharded* stands for how many slices one expert is split into. After Flexible All-to-All, tokens are immediately dispatched into a suitable format for following parallelism: each GPU gets a partial gather of tokens from All-to-All which can compute with local expert slice in tensor parallel style. In MoE combine, results on each device will be collected using global All-to-All followed by a local sum reduction. This method requires $T_{model} = O(n\text{-}sharded \cdot \Delta E \cdot C \cdot M)$ of communicate size.

All–to–all 前将每个 token 复制 n 份，n 为 sharding 数目。All–to–all 之后，每个 GPU（包含一部分的 expert 参数）可以利用一部分 tokens（这里每个 token 均为完整的）以 tensor 并行（应该指的是 GPU 上多 sm 并行计算多个 toekns）的方式算出每个 expert slice 的输出（1 个 token 对应一个输出），MoE combine 之后再 reduce（相同 token 的不同拷贝之间）

Inline Parallelism Router. Fundamentally, the combination of two strategies can well cover different expert granular-

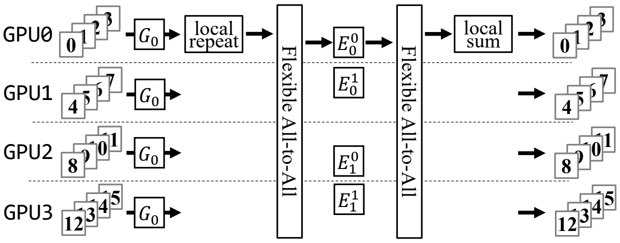


Figure 12: Switchable Expert + Model Parallelism.

Expert Parallelism: 多个 experts 在相同 GPU 上，当 experts 数目大于 GPU 数目时触发，仅限于小规模场景 (GPU 少)

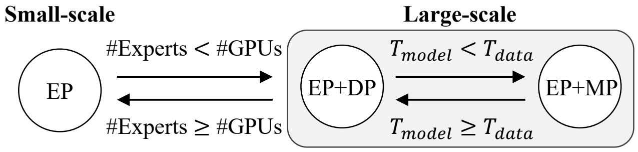


Figure 13: State Machine of Switchable Parallelism, with "EP + DP" for P1 and "EP + MP" for P2. The EP state is a special case of either "EP + DP" and "EP + MP" in small scale.

ity from small scale to large scale (as is shown in Figure 13), additionally providing the ability for an *inline parallelism router* to decide which path is better to walk along on the fly.

Inline Parallelism router uses a real-time cost function to determine optimal strategy choices. Note that P1 and P2 have theoretically equivalent local computation time, the cost function only needs to evaluate their gap in communication based on the direct calculation of communication size, so the decision can be made quickly in $O(1)$ time complexity.

Inline parallelism router: 使用一个 cost function 来实时决定最优并行方式 (考虑模型参数和 token size 的相对大小，experts 数目和 GPU 数目的相对大小)。

由于 EP+DP 和 EP+MP 在 GPU 上的计算时间相同 (这也进一步验证了 MP 中不同 expert slice 的输入都是完整的 tokens)，因此仅需考虑通信量的差异

3.3 Adaptive Pipelining

In this section, we present the design of adaptive pipelining, which comes in two folds. First, we introduce our approach to partition tokens to enable multi-stream computation-communication pipelining. Second, we demonstrate the online algorithm to search for optimal pipelining strategy for adapting to different MoE model settings and dynamic runtime capacity.

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 [13] partitions all operations in the layer. This doesn't work in MoE because it amplifies the imbalance of MoE dispatch and destroys correctness for ML features like Batch Prioritized Routing [36]. 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 14 gives 2-GPU example for data partition design in All-to-All-Expert overlapping.

传统工作 (GPipe) 将所有 layers 切成多块，这在 MoE 中加重了 dispatch 的 Imbalance (应该是不同 experts 分配的 GPU 数目不同，所以切的 stages 数目也可能不同)，甚至会影响正确性

In the forward pass, on each GPU, input of shape $(E,\Delta C,M)$ is split along dimension C into two virtual partitions of shape $(E,\Delta C/2,M)$. 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 $(\Delta E,C/2,M)$. Next, the two All-to-All outputs is 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 $(E,\Delta C,M)$.

因此仅在 all-to-all 的部分进行切分，而不是对整个 MoE layer (主要是不进行不同 experts 之间的不同 stages 切分)

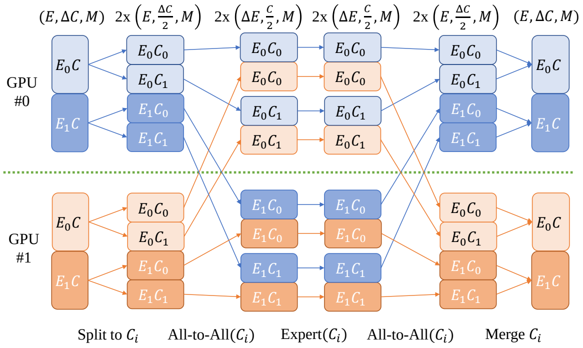

$(E,\Delta C,M)$
2x $\left(E,\frac{\Delta C}{2},M\right)$
2x $\left(\Delta E,\frac{C}{2},M\right)$
2x $\left(\Delta E,\frac{C}{2},M\right)$
2x $\left(E,\frac{\Delta C}{2},M\right)$
$(E,\Delta C,M)$
GPU #0
GPU #1
E_0C
E_1C
E_0C_0
E_0C_1
E_1C_0
E_1C_1
Split to C_i
All-to-All(C_i)
Expert(C_i)
All-to-All(C_i)
Merge C_i

Figure 14: Overview of token partition on 2-expert-2-GPU for All-to-All-Expert multi-stream overlapping. E_i means that data is sent to i-th GPU and processed by i-th expert, and C_i means that data belongs to i-th partition of capacity dimension. All-to-All and expert operations of different capacity partitions can be overlapped.

其实就是把每个 GPU 上的 token 沿 capacity dim (应该就类似于 batch dim) 切的更细，然后用相同数目的 GPU streams，在每个 stream 间进行计算和通信的 pipeline。上图中，通信 stream 仅传完 1/2 的 toekns，计算 stream 就可以开始算了，和 GPipe 搞的事情是一样的

这样做是没有额外跨 GPU 数据拷贝的

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. As a result, there is no extra data copy overhead compared with no-overlapping case.

Online pipelining strategy search. Adaptive pipelining needs to select an optimal pipelining strategy in each iteration with varied capacities f. However, it's quite difficult because the possible f distributes in large floating point number domain according to Figure 1. This leads to a large solution space and few opportunities to find the best strategy on one specefic f by running all strategies at runtime.

Expert capacity 随着 iterations 不断变化，若变得更小，则上面说的对 tokens 沿 capacity dim 再做进一步切分，进而进行 pipeline 的意义就不大了 (因为 per-expert 的 tokens 很少，计算和通信之间的 bubble 很少)

然而，由于 f 是一个浮点数，根据 workload (对应不同 f) 实时搜索最优 pipeline 的策略搜索空间太大 (f 可选取值太多，导致需要存储的 f-> 最优策略映射对太多) 这里描述成需要重搜索次数太多更合适

In order to reduce the solution space and boost the optimal strategy searching process, we propose Algorithm 2 based on one intuition: two close f should have similar workload pattern and the optimal pipelining strategy should be similar. The algorithm starts with an empty hash table used to map

对 f 进行离散化，将两个接近的 f 值映射到相同的最优 pipeline 策略上

Algorithm 2 Online Pipelining Strategy Search

```
procedure GetStrategy(capacity_factor)
    f ← capacity_factor
    if (f already in some bucket) then
        ReComputeBuckets(f)
        return GetStrategy(f)
    end if
    if (AllStrategyTried(f)) then
        return GetBestStrategy(f)
    end if
    if (AllStrategyTried(bucket)) then
        return GetBestStrategy(bucket)
    else
        return GetUntriedStrategy(bucket)
    end if
end procedure

procedure OptimizeStrategy(
    capacity_factor, strategy, measured_time)
    f ← capacity_factor
    s ← strategy
    t ← measured_time
    UpdateTriedStrategy(f, s, t)
    bucket = GetBucket(f)
    UpdateTriedStrategy(bucket, s, t)
end procedure

procedure MoEStepAndOptimzeStrategy
    f ← GetCurrentCapFactor()
    s ← GetStrategy(f)
    t ← MeasureTime(MoEStep(s))
    OptimizeStrategy(f, s, t)
end procedure
```

encountered fs to their tried strategies, an empty list used to store buckets with its tried strategies, and a pre-defined constant bucket length L. During the MoE workload, different fs will be sorted into different buckets of length L, and fs inside the same bucket will share strategies. As a result, one f can leverage strategies from others.

There are mainly three phases in MoEStepAndOptimizeStrategy for each training iteration: 1) get strategy s according to current f (GetStrategy); 2) run MoE workload with s and f, and sample its overhead as t; 3) update running time in tried strategy memo using s and t (OptimizeStrategy). During phase 1, the algorithm will first try to assign a bucket for f if not yet. After that, it will pick an optimal strategy in what f has tried at first priority, and then pick from f's bucket as second priority. If neither is the case, f will be made try some new strategy not yet tried bucket-wise and bring the performance back later by OptimizeStrategy. This ensures that each bucket would have checked all strategies with minimum repeat.

优化 pipeline 策略的算法就是对给定的 f ，遍历所有可选的策略 s 反复 profile，然后选时间性能最优的那个 s 作为结果

ReComputeBuckets is used to assign f to some bucket. It will add it into the known f list, sort the list, and re-compute buckets for all known fs. This is done by greedily putting new f into existing L-sized bucket until it exceeds the bucket size L, where current bucket is settled and a new bucket starting from the f to be added is created. The optimal strategy list of each new bucket will be constructed from fs belonging to it, with time being normalized by lowest f in the new bucket.

The overall strategy decision and optimization process takes $O(1)$ time if f is known, for indexing f in the hash table and access its corresponding bucket. When f is new, it takes $O(\log(M))$ in average where M buckets are binary searched, and $O(N\log(N))$ in worst case where buckets are re-generated, and N existing capacity factors are sorted.

3.4 Two-dimensional Hierarchical All-to-All

To tackle the inefficiencies in Section 2.4, 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.

为了处理当 scale out 到极大规模 GPU num 的时候，GPU 间传输量很低的问题

Challenge. *The overhead of merging small messages* is the key challenge to make the chunk aggregation time constant and do not increase even in a very large scale. 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, which is equivalent to perform $\frac{S}{n}$ size intra-node All-to-All $\frac{n}{m}$ times, as illustrated in Figure 15, 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 $O(\frac{n}{m})$ 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 $n = 2048$.

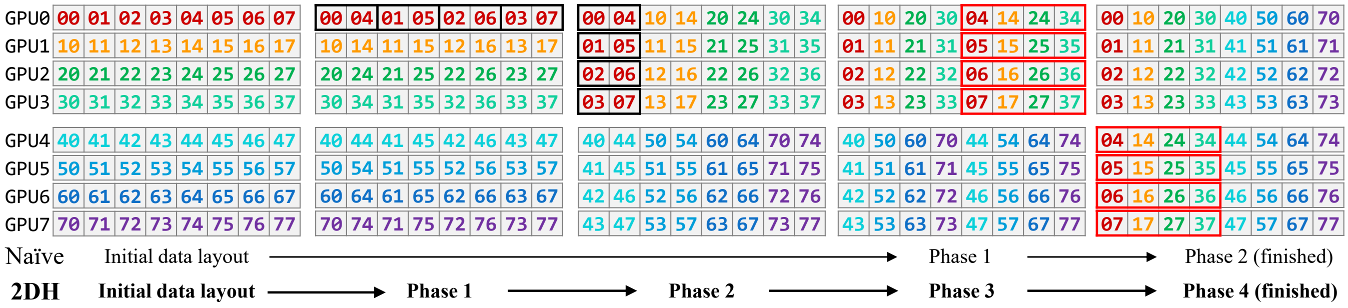
S 是模型大小，n 是 GPUs 总数，m 是每个 node 内的 GPUs 数目。根据 all-reduce 的定义，每个 data chunk 的大小为 S / n。

在 intra-node all-to-all 中，每两个 GPU 之间仅需交换 S / m (与 n 无关 -> 理论上通信延迟不会随着 scale out 添加更多 nodes 而变化)。但其实，由于在每个 GPU 上进行了 n / m 次非连续的内存访问 (block 0 一次，block 4 一次，在 intra-node all-to-all 时)，因此通信延迟实际上也会增加

Algorithm. To relax the All-to-All performance downgrade due to the small message size and the large number of connections, we propose *two-dimensional hierarchical (2DH) All-to-All algorithm* that merges small messages over multiple connections into a large message over a single connection. We leverage high-bandwidth intra-node links (NVLink [30] for NVIDIA GPUs or Infinity Fabric Link (xGMI) [2] for AMD GPUs) to aggregate small messages in each node.

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 15 illustrates all phases of 2DH All-to-All in order. In-

注意，naive 方法是先 intra-node 直接 all-to-all allreduce，每次发一个 S/n 的 data chuck (GPUs 间发送 data chunk 是线性的，即发完 chunk 0 再发 chunk 1)，一对 GPUs 间总共发 n / m 个 data chunks，发的时间更长，利用率低



GPU0	00 01 02 03 04 05 06 07	00 04 01 05 02 06 03 07	00 04 10 14 20 24 30 34	00 10 20 30 04 14 24 34	00 10 20 30 40 50 60 70
GPU1	10 11 12 13 14 15 16 17	10 14 11 15 12 16 13 17	01 05 11 15 21 25 31 35	01 11 21 31 05 15 25 35	01 11 21 31 41 51 61 71
GPU2	20 21 22 23 24 25 26 27	20 24 21 25 22 26 23 27	02 06 12 16 22 26 32 36	02 12 22 32 06 16 26 36	02 12 22 32 42 52 62 72
GPU3	30 31 32 33 34 35 36 37	30 34 31 35 32 36 33 37	03 07 13 17 23 27 33 37	03 13 23 33 07 17 27 37	03 13 23 33 43 53 63 73
GPU4	40 41 42 43 44 45 46 47	40 44 41 45 42 46 43 47	40 44 50 54 60 64 70 74	40 50 60 70 44 54 64 74	04 14 24 34 44 54 64 74
GPU5	50 51 52 53 54 55 56 57	50 54 51 55 52 56 53 57	41 45 51 55 61 65 71 75	41 51 61 71 45 55 65 75	05 15 25 35 45 55 65 75
GPU6	60 61 62 63 64 65 66 67	60 64 61 65 62 66 63 67	42 46 52 56 62 66 72 76	42 52 62 72 46 56 66 76	06 16 26 36 46 56 66 76
GPU7	70 71 72 73 74 75 76 77	70 74 71 75 72 76 73 77	43 47 53 57 63 67 73 77	43 53 63 73 47 57 67 77	07 17 27 37 47 57 67 77

Figure 15: 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.

2DH All-to-All 算法：
(1) 在每个 GPU 内用 stride memory copy (可以消除非连续内存访问的延迟) 来高效地将非连续但目标 GPU 相同的 data chunks align 到一起；
(2) 在相同 node 的 GPUs 间进行 intra-node all-to-all，每对 GPUs 间的通信量为 S / m，交换一个 aligned data chunks group；
(3) 同样在每个 GPU 上将非连续但目标 node 相同的 data chunks 用 stride memory copy align 到一起；
(4) 在 nodes 间将第二次的 aligned data chunks group 进行 inter-node all-to-all，每对 nodes 间的通信量为 S / (n / m)，和 naive 方法相同。

因此，该方法仅提高了 intra-node 的网络利用率，即 high-bandwidth intra-node links (NVLink [30] for NVIDIA GPUs or Infinity Fabric Link (xGMI) [2] for AMD GPUs)。

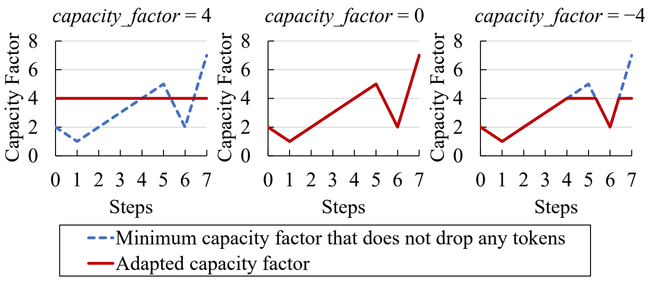


Figure 16: Examples of dynamic capacity factor adaptation when *capacity_factor* is given as 4, 0, and −4, respectively.

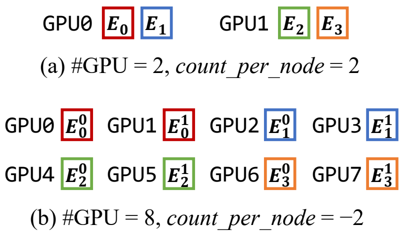
stead of performing intra-node All-to-All from the beginning like the naïve local aggregation, we first align chunks that share the same local destination GPU via stride memory copies (phase 1) and then conduct intra-node All-to-All (phase 2). In the following phase, again, we align chunks that share 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.

4 Implementation

This section presents details of key features and optimizations provided by TUTEL.

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.



(a) #GPU = 2, *count_per_node* = 2

(b) #GPU = 8, *count_per_node* = −2

Figure 17: Examples of distributing four global experts w.r.t the value of *count_per_node*.

***Dynamic* Top-*ANY* MoE Gating.** TUTEL supports top-ANY routing that most of existing frameworks do not support. The k value can be changed dynamically at runtime as well, i.e., different iterations of one MoE layer can 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.** TUTEL supports adjusting the capacity factor dynamically at every iterations. As illustrated in Figure 16, the adjustment behavior is controlled by $capacity\_factor = x$ argument 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, TUTEL performs the same automatic adaptation like when x is zero, except that $-x$ is set as the upper bound of capacity factor, i.e., any exceeding value will be adapted to $-x$.

Handy Control of Parallel Distribution. While TUTEL automatically adapts parallelism as described in Section 3.2, it also supports transparent control of parallelism by users. TUTEL provides an easy-to-use interface for the control, $count\_per\_node = x$, an argument passed to MoE layer API. Even with a single argument, TUTEL automatically constructs a desired expert distribution accordingly, which covers diverse expert distributions including multiple experts in a single GPU or each expert split across multiple GPUs. As shown in Figure 17, if x is a positive integer, each GPU manages x local experts. Otherwise, each expert is parallelized across

```
# Tensor shapes: logits(T,E)
gate_probs = softmax(logits)
# Tensor shapes: gate_probs(T,E), idxs(T,), scores(T,)
idxs, scores = top_k(gate_probs)
# Tensor shapes: locations(T,)
locations = compute_location(idxs)
# Tensor shapes: locations(T,), locations1(T,ΔC)
locations1 = one_hot(locations, num_classes=ΔC)
# Tensor shapes: gate_probs(T,E), combine(T,E,ΔC)
combine = einsum("TE,TC->TEC", gate_probs, locations1)
# Tensor shapes: dispatch_input(E,ΔC,M),moe_input(T,M)
dispatch_input = einsum(
    "TEC,TM->ECM", bool(combine), moe_input)
```

(a) Dense implementation.

```
# Tensor shapes: logits(T,E)
gate_probs = softmax(logits)
# Tensor shapes: gate_probs(T,E), idxs(T,), scores(T,)
idxs, scores = top_k(gate_probs)
# Tensor shapes: locations(T,)
locations = compute_location(idxs)
# Tensor shapes:
# dispatch_input(E,ΔC,M), moe_input(T,M)
dispatch_input = zeros((E,ΔC,M))
for t in [0, 1, ..., T-1]:
    # Broadcast multiplication
    dispatch_input[idxs[t]][locations[t]] = \
        bool(scores[t]) * moe_input[t]
```

(b) Sparse implementation.

Figure 18: 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`).

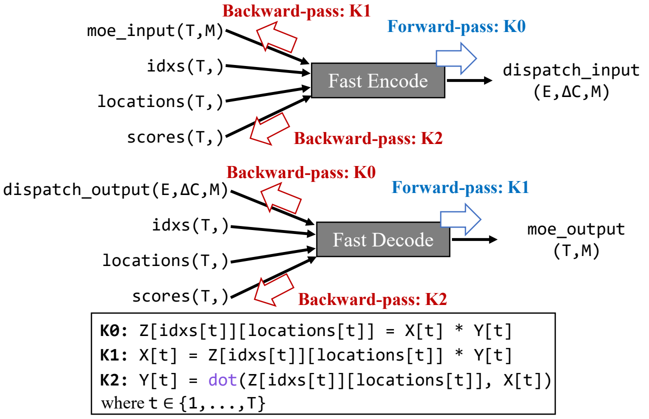
$-x$ GPUs where each GPU handles $1/(-x)$ of the entire input data of a single expert. *count_per_node* only deals with throughput while keeping the training algorithm unchanged.

Different Devices and Data Types. TUTEL supports different computation devices including NVIDA GPU, AMD GPU, and CPU. It also has all native data type support on different devices: FP64/FP32 for CPU backend, and FP64/FP32/FP16/BF16 for GPU backends.

Compatible with Any DNN Architectures. While this paper evaluates TUTEL only with a Transformer-based model architecture, it can be used with any other DNN architectures such as MLP, CNN, or LSTM.

4.2 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 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 [18] and implemented in Fairseq [33]. For instance, Figure 18a 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 multiplication and addition. TUTEL addresses this by a sparse implementation as shown in Figure 18b. Given that T is the number of input tokens per expert, while the time complexity of the dense version is $O(T \cdot E \cdot \Delta C \cdot M)$, the one of the sparse version is only $O(T \cdot k \cdot M)$, where $T \cdot k = E \cdot \Delta C$ in most cases. This indicates that the sparse version has only $1/T$ of time complexity than the dense version.



```
K0: Z[idxs[t]][locations[t]] = X[t] * Y[t]
K1: X[t] = Z[idxs[t]][locations[t]] * Y[t]
K2: Y[t] = dot(Z[idxs[t]][locations[t]], X[t])
where t ∈ {1,...,T}
```

Figure 19: 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,M), $(T,)$, and $(E,\Delta C,M)$, respectively. `idxs` and `locations` have no backward-pass computation as they are not trainable inputs.

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.[4]

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 19. 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, Blelloch 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 24. It greatly saves GPU memory as well. As shown in Table 4, in most cases, it achieves 20% $\sim$ 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.

[4] Even the sparsity support by the latest hardware (e.g., 3rd-generation Tensor Cores) cannot work efficiently as it only supports fine-grained sparsity, while our sparse computation belongs to coarse-grained sparsity [26].

Algorithm 3 Two-Dimensional Hierarchical (2DH) All-to-All

```
procedure StrideMemcpy(output, input, chunksize, row, col)
    for i = 0; i < row × col; i++ do
        j ← i % row × col + i / col
        output[j × chunksize : (j+1) × chunksize] ← input[i × chunksize : (i+1) × chunksize]
    end for
end procedure
procedure All2All_2DH(output, input)
    // step 1: intra-node All-to-All
    strideMemcpy(buffer, input, chunksize, ngpus_per_node, nnodes)
    for g = 0; g < ngpus_per_node; g++ do                                        ▷ ncclGroupStart
        loc ← g × nnodes × chunksize, peer ← g + node_rank × ngpus_per_node
        ncclSend(buffer[loc], nnodes × chunksize, datatype, peer, comm)
        ncclRecv(output[loc], nnodes × chunksize, datatype, peer, comm)
    end for                                                                       ▷ ncclGroupEnd
    strideMemcpy(buffer, output, chunksize, nnodes, ngpus_per_node)
    // step 2: inter-node All-to-All
    for n = 0; n < nnodes; n++ do                                                 ▷ ncclGroupStart
        loc ← n × ngpus_per_node × chunksize, peer ← local_rank + n × ngpus_per_node
        ncclSend(buffer[loc], ngpus_per_node × chunksize, datatype, peer, comm)
        ncclRecv(output[loc], ngpus_per_node × chunksize, datatype, peer, comm)
    end for                                                                       ▷ ncclGroupEnd
end procedure
```

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 4: GPU memory cost for single MoE layer. (Static Settings: M = V = 4096, top-k = 2, ΔE = 2)

4.3 Optimized 2DH All-to-All with MSCCL

Implementation using NCCL APIs. We implement 2DH All-to-All algorithm using NCCL's `ncclSend` and `ncclRecv` APIs, as described in Algorithm 3. It consists of two steps. The first step corresponds to phase $1 \sim 3$ in Figure 15 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 15, which is inter-node All-to-All and its latency relies on n/m instead of n as local chunks are already merged.

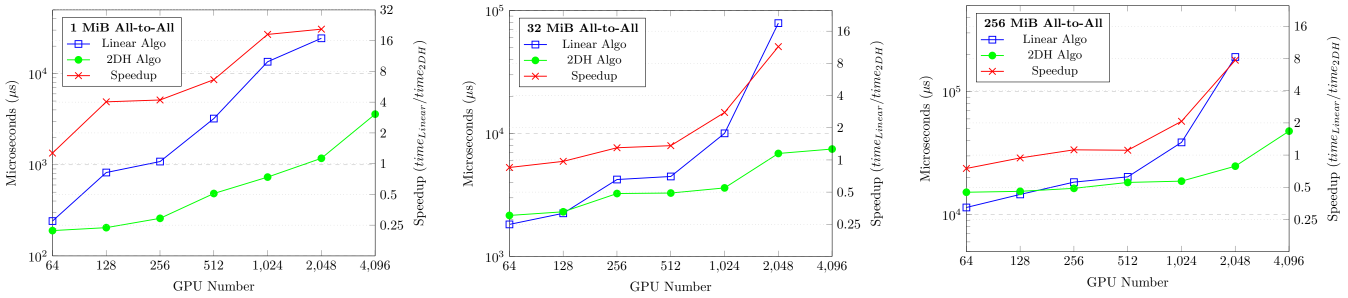
Optimization via MSCCL. Implementation using NCCL APIs requires extra synchronization barriers between different phases in 2DH All-to-All and may cause throughput degradation. In order to achieve better performance, we leverage MSCCL by describing the 2DH algorithm in a domain specific language (DSL) and optimizing with the compiler [8]. The custom compiler also leverages LL128 protocol [32] 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 [30] 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 [16], 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 networking hierarchy.

5 Evaluation

Testbed. If not specified, all experiments use Azure Standard_ND96amsr_A100_v4 VMs [4] . Each VM is equipped with 8× NVIDIA A100 SXM 80GB GPUs and 8× 200 Gbps HDR InfiniBand, backed by 96× 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 [29] and NCCL RDMA SHARP plugin [25] are used for communication when scaling out. Micro-benchmarks use up to 4,096 A100 GPUs (512 VMs) for scaling.

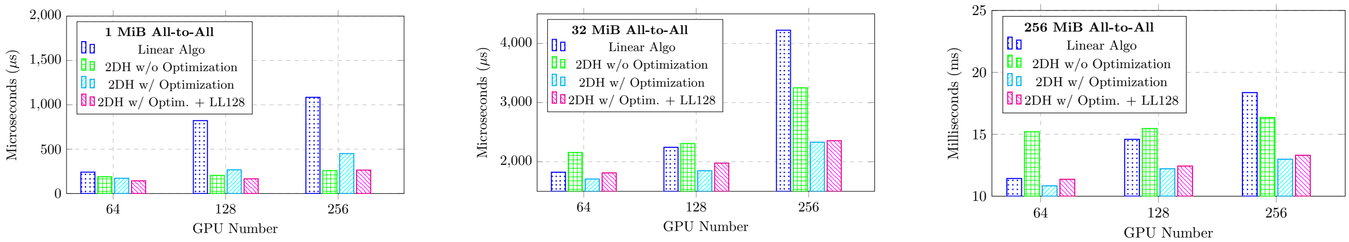


(a) All-to-All 1 MiB

(b) All-to-All 32 MiB

(c) All-to-All 256 MiB

Figure 20: Comparison between linear and 2DH All-to-All algorithms with various sizes in NCCL.



(a) All-to-All 1 MiB

(b) All-to-All 32 MiB

(c) All-to-All 256 MiB

Figure 21: Comparison between NCCL and optimized implementation [8] running 2DH All-to-All algorithm.

5.1 Micro-benchmarks

5.1.1 2DH All-to-All

We benchmark `alltoall_perf` in nccl-tests [28] to measure the performance and correctness of All-to-All operations. 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 [28] 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 in Figure 20, 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 [8]. As illustrated in Figure 21, 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.

5.1.2 Adaptive Parallelism Switching

We evaluate adaptive parallelism switching with different MoE model settings on single node. For comparison, we measure the throughput improvement for adaptive parallelism switching against static parallelism. Table 5a shows the improvements based on dynamic f while other settings are fixed. The adaptive parallelism switches to P2 for small f and P1 for large f, which makes up to 23.1% improvement if P2 is chosen statically during training.

Table 5b evaluates different model settings for adaptive parallelism switching. Adaptive method tends to choose P2 for large fflayer hidden_size and would turn to P1 for large tokens/step. The number of global experts also influences the preference of adaptive method: Less global experts require larger *n-shard* pieces to slice an expert, which has bad impact on P2's communication, while it has insignificant impact on P1. For the last setting, it is evaluated based on a hybrid f varies from 1 to 16, adaptive method can achieve up to 5.8% $\sim$ 8.1% improvement against both P1 and P2 simultaneously,

$E2,S2K,V8K$	$f1$	$f2$	$f4$	$f8$	$f16$
P1	19.2%	2.3%	0%	0%	0%
P2	0%	0%	9.7%	18.6%	23.1%

(a) Adaptive parallelism improvement on different capacity-factors f.

Settings	P1	P2
$f1,E4,S1K,V4K$	29.7%	0%
$f1,E4,S1K,V8K$	47.8%	0%
$f1,E2,S16K,V2K$	0%	58.3%
$f1,E2,S32K,V2K$	0%	74.7%
$f1,E4,S4K,V8K$	12.8%	0%
$f1,E1,S4K,V8K$	0%	10.7%
$f1\sim16,E4,S2K,V8K$	8.1%	5.8%

(b) Adaptive parallelism improvement on different settings.

Table 5: Adaptive parallelism switching improvement on different settings against the wort case. fa,Eb,Sc,Vd stands for $f=a$, $E=b$, $V=d$, tokens/step = c. Static settings: W = 8, M = 2K.

Parameters	Settings
samples/step	8 / 16 / 32
tokens/sample	512 / 1,024 / 2,048
M	1,024 / 2,048 / 4,096
V	1,024 / 2,048 / 4,096
ΔE	0.5 / 1 / 2

Table 6: Typical single MoE model settings.

due to its flexibility to align with preferred parallelism in different iterations.

5.1.3 Adaptive Pipelining

We evaluate adaptive pipelining on 243 typical MoE model settings on different scale (16 ∼ 256 GPUs). The detailed MoE settings are described in Table 6. As the comparison, we also measure different static pipelining methods considering different degrees {1, 2, 4, 8} and different All-to-All algorithms (Linear and 2DH).

Table 7a shows average improvement on these 243 models. Compared with baseline solution (pipelining degree 1) and Linear All-to-All), adaptive piplining achieves 9% ∼ 101% improvement in average. Compared with different static strategies, it also can achieve 1% ∼ 107% improvement in average. Besides, adaptive piplining achieves significant improvement and avoids performance regression in the worst case, which shows 23% ∼ 599% improvement in Table 7b.

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 22, adaptive pipelining always chooses the best strategy, and it can achieve up to 30% improvement with $f=4$ and up to 67% improvement with $f=16$, compared with baseline (pipelining degree 1).

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

(b) Adaptive pipelining improvement over the worst case.

Table 7: Adaptive pipelining improvements.

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 = 16,384, f = 1, M = 2,048, V = 2,048, ΔE = 2, top-k = 2. We add TUTEL features once at a time to study where the major gain is from, Fairseq [33] is used as baseline in the experiments.

The following explains each curve in Figure 23 in order. ① Fairseq MoE Baseline. ② TUTEL Kernel + Linear All-to-All. TUTEL kernel optimizations deliver a huge gain at a small scale (3.52× on 16 GPUs), while the gain becomes small at a large scale (1.04× on 2,048 GPUs). The detailed gains from using TUTEL kernels over Fairseq is shown in Figure 24. ③ TUTEL Kernel + Adaptive Pipelining. Adaptive pipelining will choose the optimal All-to-All algorithm and pipelining degree dynamically, which delivers a significant gain because of using 2DH algorithm at a large scale (4.25× on 2,048 GPUs). ④ TUTEL Kernel + Adaptive Pipelining + Flexible All-to-All. Flexible All-to-All delivers gains at large scales starting from 256 GPUs, e.g., 1.24× on 2,048 GPUs compared with not using it. ⑤ TUTEL Kernel + Adaptive Pipelining + Flexible All-to-All + Adaptive Parallelism Switching. It delivers the full set of adaptivity in TUTEL. Compared with the baseline, TUTEL finally delivers **4.96×** and **5.75×** speedup on 16 GPUs and 2,048 GPUs, respectively. For computation-communication breakdown, ⑥ shows only

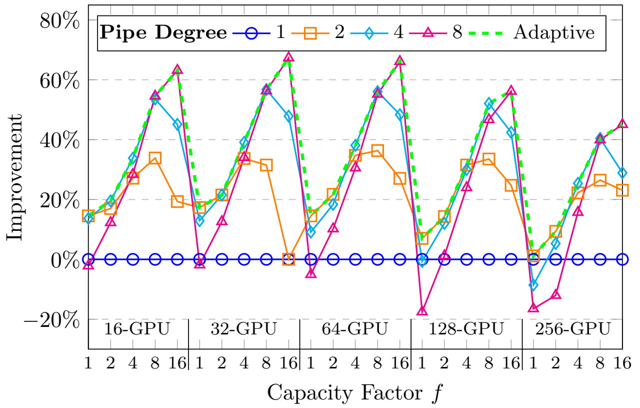


Figure 22: Adaptive pipelining improvement on dynamic workload. In the model, tokens/step = 4,096, M = 4,096, V = 4,096, ΔE = 2.

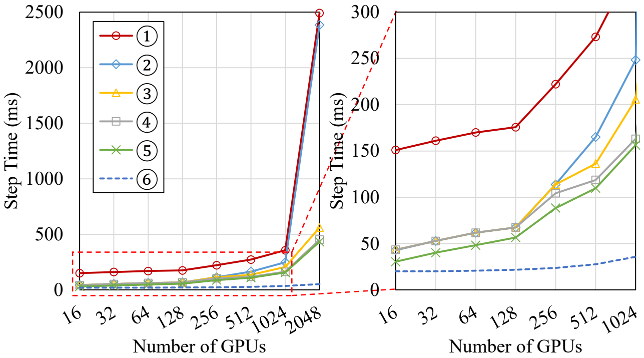
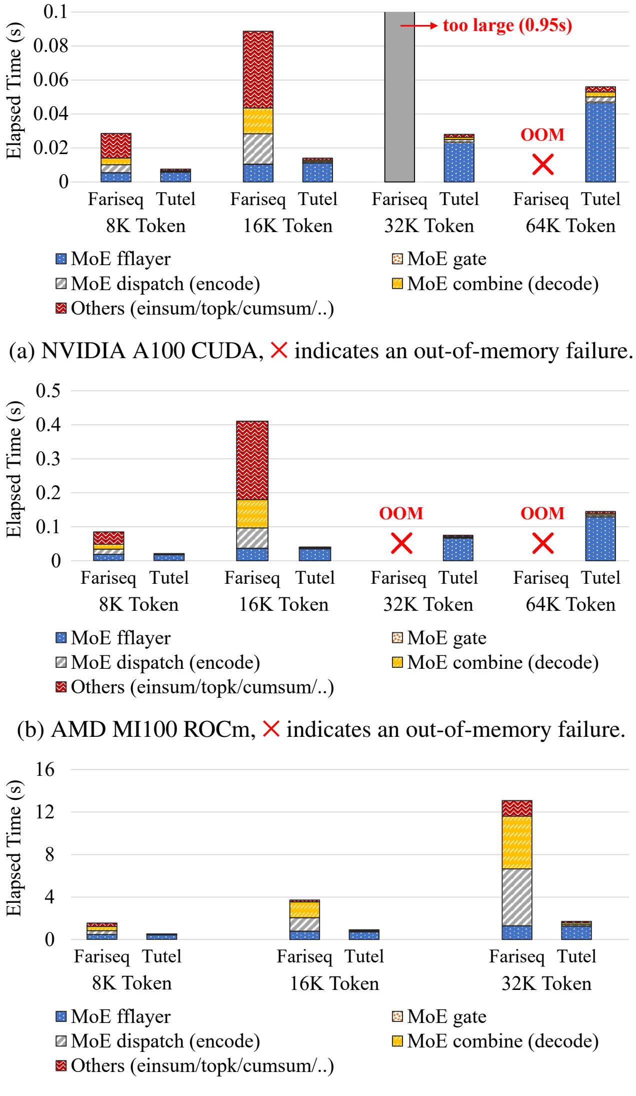


Figure 23: Single MoE layer improvement breakdown.

the computation overhead of the complete TUTEL (excluding the portion overlapped with communication). The computation overhead slightly increases when we scale out due to more theoretical computation required from gating function, as the number of total experts increases together with the number of GPUs.

5.3 On Real-World Computer Vision Problems using 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 [22, 23], 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 hyper-parameters are: $E = 32$, top-$k = 1$, capacity factor $f = 1.0$.



(a) NVIDIA A100 CUDA, ✕ indicates an out-of-memory failure.

(b) AMD MI100 ROCm, ✕ indicates an out-of-memory failure.

(c) AMD CPU EPYC 7V12 Processor

Figure 24: Kernel computation breakdown between TUTEL and public Fairseq MoE as baseline

5.3.1 Experiment Setup

Pre-training and Down-stream Computer Vision Tasks. We follow [23] 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 [36]. 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 [21]. The pre-trained models are fine-tuned on the

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

Table 8: Comparing the training and inference speeds (images per second) of SwinV2-MoE using Fairseq and TUTEL.

Method	IN-22K acc@1	IN-1K/ft acc@1	IN-1K/5-shot acc@1	COCO (AP) 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 9: Comparing the pre-training and fine-tuning accuracy between the sparse SwinV2-MoE-B model and its dense counterpart.

COCO object detection training set using a cascade mask R-CNN framework [23], and box/mask AP on the validation set is reported.

5.3.2 Experiment Results

Speed Comparison. Table 8 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. In training, each iteration step is speedup by $1.14\times \sim 1.55\times$. In inference, the speedup is $1.95\times \sim 2.11\times$.

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 9. 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.

How to do fine-tuning on COCO object detection? Previous MoE models on computer vision only perform experiments using image classification tasks [36]. 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.

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.

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.

5.3.3 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 [9, 36].

Comparison of Routing Algorithms and Capacity Factors. Figure 25 compares the routing methods with and without batch prioritized routing (BPR) [36]. 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 [36].

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 $f = 1.0$.

5.3.4 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 framework for algorithmic innovation. One attempt is a new cosine router that hopes to improve numerical stability with increased model size, inspired by [22]:

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

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; τ 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,

Method	E	k	f	$\#param$	$\#param_{act}$	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 [22]. 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 192×192. We report the top-1 accuracy and final trainning 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.

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 [6].

6 Related Work

MoE Frameworks. While GShard [18] provides a computation logic that ensures algorithmic correctness of MoE, several popular MoE frameworks [17, 33] follow the same logic but perform poorly at a large scale. FastMoE/FaterMoE [12] proposes different gating algorithms that are not computationally equivalent with GShard. Furthermore, proposed *shadow expert* and *smart schedule* in FasterMoE introduce several restrictions on dataset, expert portage cost, and memory for caching. TUTEL keeps the same computation logic as GShard and achieves deterministic gain on typical environments and hyper-parameter settings, which adapts MoE frameworks to exa-scale without harming algorithmic results.

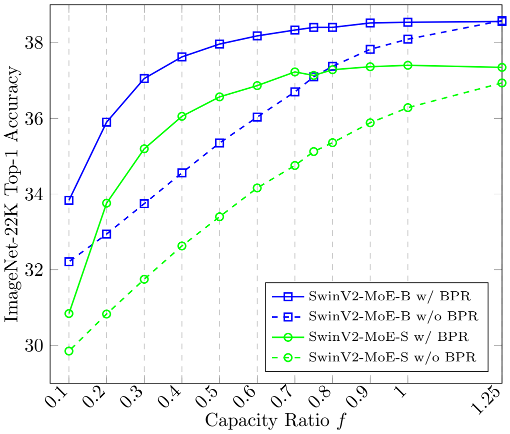


Figure 25: ImageNet-22K top-1 accuracy w.r.t inference capacity factor. "w/ BPR" indicates training with batch prioritized routing while "w/o BPR" not. All models are trained on the ImageNet-22K dataset with $E = 32$, $k = 1$, $f = 1.25$ and an input resolution of 192×192 for 90 epochs.

All-to-All Collective Communication. NCCL is the state-of-the-art collective communication library on NVIDIA

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 ($E = 32$, $k = 1$, $f = 1.25$).

GPUs and InfiniBand, while there is also a forked version, RCCL [3], on AMD GPUs that shares similar implementations. The recent NCCL 2.12 version introduces a feature called *PXN* [24] that avoids cross-rail communication by leveraging intra-node communication. While PXN delivers a similar gain as 2DH All-to-All in terms of intra-node chunk aggregation, it still does not address the challenge of small message merging overhead in Section 3.4. PXN claims $\sim 2.5\times$ speedup for 1 MiB size at 1,024 GPUs [24], while our algorithm shows ten to twenty times of speedup for the same size at 1,024 or 2,048 GPUs. Please note that PXN is a parallel work with this work, and it is yet early to deliver an extensive comparison in this paper.

7 Conclusion

Mixture-of-Experts (MoE) is the key technology to pre-train trillion-plus parameter models and has proven its promising performance with better model qualities at exa-scale. 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× 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.

References

[1] DeepSpeed. https://www.deepspeed.ai/, 2022. [Online; accessed Mar 2022].

[2] AMD. Introducing AMD CDNA 2 Architecture. https://www.amd.com/system/files/documents/amd-cdna2-white-paper.pdf, 2022. [Online; accessed Feb 2022].

[3] AMD. ROCm Communication Collectives Library (RCCL). https://github.com/ROCmSoftwarePlatform/rccl/tree/2.10.3, 2022. [Online; accessed Feb 2022].

[4] Microsoft Azure. NDm A100 v4-series - Azure Virtual Machines. https://docs.microsoft.com/en-us/azure/virtual-machines/ndm-a100-v4-series, 2022. [Online; accessed Feb 2022].

[5] Jehoshua Bruck, Ching-Tien Ho, Shlomo Kipnis, Eli Upfal, and Derrick Weathersby. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Transactions on parallel and distributed systems*, 8(11):1143–1156, 1997.

[6] Zewen Chi, Li Dong, Shaohan Huang, Damai Dai, Shuming Ma, Barun Patra, Saksham Singhal, Payal Bajaj, Xia Song, and Furu Wei. On the representation collapse of sparse mixture of experts, 2022.

[7] Aidan Clark, Diego de Las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake A. Hechtman, Trevor Cai, Sebastian Borgeaud, George van den Driessche, Eliza Rutherford, Tom Hennigan, Matthew Johnson, Katie Millican, Albin Cassirer, Chris Jones, Elena Buchatskaya, David Budden, Laurent Sifre, Simon Osindero, Oriol Vinyals, Jack W. Rae, Erich Elsen, Koray Kavukcuoglu, and Karen Simonyan. Unified scaling laws for routed language models. *CoRR*, abs/2202.01169, 2022.

[8] Meghan Cowan, Saeed Maleki, Madanlal Musuvathi, Olli Saarikivi, and Yifan Xiong. MSCCL: microsoft collective communication library. *CoRR*, abs/2201.11840, 2022.

[9] Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathy Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V. Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. Glam: Efficient scaling of language models with mixture-of-experts. *CoRR*, abs/2112.06905, 2021.

[10] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *CoRR*, abs/2101.03961, 2021.

[11] Roy Frostig, Matthew Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. In *The Conference on Systems and Machine Learning (SysML)*, 2018.

[12] Jiaao He, Jidong Zhai, Tiago Antunes, Haojie Wang, Fuwen Luo, Shangfeng Shi, and Qin Li. 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.

[13] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Xu Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[14] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Comput.*, 3(1):79–87, 1991.

[15] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020.

[16] John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. IEEE, 2008.

[17] Young Jin Kim, Ammar Ahmad Awan, Alexandre Muzio, Andrés Felipe Cruz-Salinas, Liyang Lu, Amr Hendy, Samyam Rajbhandari, Yuxiong He, and Hany Hassan Awadalla. Scalable and efficient moe training for multitask multilingual models. *CoRR*, abs/2109.10465, 2021.

[18] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *CoRR*, abs/2006.16668, 2020.

[19] Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. BASE layers: Simplifying training of large, sparse models. In Marina Meila and Tong Zhang, editors, *Proceedings of the International Conference on Machine Learning (ICML)*, 2021.

[20] Junyang Lin, An Yang, Jinze Bai, Chang Zhou, Le Jiang, Xianyan Jia, Ang Wang, Jie Zhang, Yong Li, Wei Lin, Jingren Zhou, and Hongxia Yang. M6-10T: A sharing-delinking paradigm for efficient multi-trillion parameter pretraining. *CoRR*, abs/2110.03888, 2021.

[21] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

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

[23] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[24] Karthik Mandakolathur and Sylvain Jeaugey. Doubling all2all Performance with NVIDIA Collective Communication Library 2.12. https://developer.nvidia.com/blog/doubling-all2all-performance-with-nvidia-collective-communication-library-2-12/, 2022. [Online; accessed Mar 2022].

[25] Mellanox. RDMA and SHARP Plugins for NCCL Library. https://github.com/Mellanox/nccl-rdma-sharp-plugins, 2022. [Online; accessed Feb 2022].

[26] NVIDIA. NVIDIA A100 Tensor Core GPU Architecture – Unprecedented Acceleration at Every Scale. Whitepaper, 2020.

[27] NVIDIA. How does NCCL decide which algorithm to use? https://github.com/NVIDIA/nccl/issues/457, 2022. [Online; accessed Apr 2022].

[28] NVIDIA. NCCL Tests. https://github.com/NVIDIA/nccl-tests, 2022. [Online; accessed Feb 2022].

[29] NVIDIA. NVIDIA Collective Communications Library (NCCL). https://github.com/NVIDIA/nccl/tree/v2.10.3-1, 2022. [Online; accessed Feb 2022].

[30] NVIDIA. NVLink & NVSwitch: Fastest HPC Data Center Platform. https://www.nvidia.com/en-us/data-center/nvlink/, 2022. [Online; accessed Feb 2022].

[31] 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, 2022. [Online; accessed Apr 2022].

[32] NVIDIA. What is LL128 Protocol? https://github.com/NVIDIA/nccl/issues/281, 2022. [Online; accessed Feb 2022].

[33] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[35] Jelena Pjesivac-Grbovic. Towards automatic and adaptive optimizations of mpi collective operations. 2007.

[36] Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. *CoRR*, abs/2106.05974, 2021.

[37] Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, and Jason Weston. Hash layers for large sparse models. *CoRR*, abs/2106.04426, 2021.

[38] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

[39] Or Sharir, Barak Peleg, and Yoav Shoham. The cost of training NLP models: A concise overview. *CoRR*, abs/2004.08900, 2020.

[40] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[41] Marc Snir, William Gropp, Steve Otto, Steven Huss-Lederman, Jack Dongarra, and David Walker. *MPI–the Complete Reference: the MPI core*, volume 1. MIT press, 1998.

[42] Rajeev Thakur and Alok Choudhary. All-to-all communication on meshes with wormhole routing. In *Proceedings of 8th International Parallel Processing Symposium*, pages 561–565. IEEE, 1994.

[43] An Yang, Junyang Lin, Rui Men, Chang Zhou, Le Jiang, Xianyan Jia, Ang Wang, Jie Zhang, Jiamang Wang, Yong Li, Di Zhang, Wei Lin, Lin Qu, Jingren Zhou, and Hongxia Yang. Exploring sparse expert models and beyond. *CoRR*, abs/2105.15082, 2021.

[44] Seniha Esen Yüksel, Joseph N. Wilson, and Paul D. Gader. Twenty years of mixture of experts. *IEEE Trans. Neural Networks Learn. Syst.*, 23(8):1177–1193, 2012.