PipeMoE: Accelerating Mixture-of-Experts through Adaptive Pipelining

Shaohuai Shi[†], Xinglin Pan[‡], Xiaowen Chu[§‡*], Bo Li[¶]
[†]School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen
[‡]Department of Computer Science, Hong Kong Baptist University
[§]Data Science and Analytics Thrust, The Hong Kong University of Science and Technology (Guangzhou)
[¶]Department of Computer Science and Engineering, The Hong Kong University of Science and Technology
shaohuais@hit.edu.cn, csxlpan@comp.hkbu.edu.hk, xwchu@ust.hk, bli@cse.ust.hk

IEEE INFOCOM 2023 - IEEE Conference on Computer Communications | 979-8-3503-3414-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOM53939.2023.10228874

***Abstract*—Large models have attracted much attention in the AI area. The sparsely activated mixture-of-experts (MoE) technique pushes the model size to a trillion-level with a sub-linear increase of computations as an MoE layer can be equipped with many separate experts, but only one or two experts need to be trained for each input data. However, the feature of dynamically activating experts of MoE introduces extensive communications in distributed training. In this work, we propose PipeMoE to adaptively pipeline the communications and computations in MoE to maximally hide the communication time. Specifically, we first identify the root reason why a higher pipeline degree does not always achieve better performance in training MoE models. Then we formulate an optimization problem that aims to minimize the training iteration time. To solve this problem, we build performance models for computation and communication tasks in MoE and develop an optimal solution to determine the pipeline degree such that the iteration time is minimal. We conduct extensive experiments with 174 typical MoE layers and two real-world NLP models on a 64-GPU cluster. Experimental results show that our PipeMoE almost always chooses the best pipeline degree and outperforms state-of-the-art MoE training systems by 5%-77% in training time.**

***Index Terms*—Distributed Deep Learning; Communication-Efficient Training; Mixture-of-Experts; Pipelining**

I. Introduction

With the development of large models in deep learning (DL), it tends to achieve higher performance in generalization with increased model size (e.g., GPT-3 [1] with 175 billion parameters, PaLM [2] with 540 billion parameters, Switch Transformer [3] with 1.6 trillion parameters, etc.). However, extremely large models take a huge cost of computing resources to train. For example, training a PaLM model requires around 2.5×10^{24} FLOPs and it takes 64 days on a cluster with 6,144 Google TPUv4 chips [2]. Such a numerous amount of computing consumption makes it extremely expensive to further increase the model size. In recent years, sparsely-activated Mixture-of-Experts (MoE) [4,5] layers are widely exploited to increase the model size to improve its generalization ability while requiring only a sub-linear increase of computations [3,6]–[8]. Compared to traditional dense layers which should be computed for every training sample, an MoE layer includes multiple dense layers (called experts) but only activates a few experts to compute the output for each training sample. For example, Switch Transformer [3] scales the size of parameters from several billion to 1.5 trillion with 15 MoE layers, each of which has 2048 experts. Thus, the model size can be easily scaled to a trillion-level while only requiring a relatively small amount of extra computational cost. Yet, when training MoE models on dense-GPU clusters, it still has some critical performance issues that limit the scalability of the training systems [9,10].

Specifically, in every iteration of training MoE layers, the input data should be dynamically distributed to different experts for computation, but the experts are typically located on different GPUs as one GPU cannot store all experts [6]. It means that the input data should be transferred to particular GPUs, which is generally implemented by an all-to-all collective communication to distribute the data (*dispatch*) to different GPUs; and the results of experts in different GPUs are then collected by another all-to-all operation (*combine*) [6]. The communication time of all-to-all operations is critical to the overall training performance. As reported in [6,10], the all-to-all communication time occupies 30%-60% of the overall time of the MoE layers on high-end dense-GPU or Google TPU clusters.

Existing distributed DL systems try to optimize the performance of MoE training in three orthogonal directions: 1) design load-balancing routing functions [3,11,12] to make the computation workloads of distributed GPUs more balanced, 2) design efficient all-to-all algorithms [8,10,13]–[16] to reduce the communication complexity, and 3) design computation and communication task scheduling algorithms [9,10] to pipeline communication tasks with computing tasks so that some communication costs can be hidden.

In this paper, we focus on the third direction by proposing an adaptive pipelining algorithm, PipeMoE, for efficient distributed training of large MoE models. Pipelining in MoE needs to partition the input data into multiple chunks to enable the overlap of communication and computing tasks on different chunks [9,10]. Intuitively, a larger number of chunks has a higher pipeline degree (i.e., the number of communication tasks can be pipelined), and thus it could achieve a shorter time (§II-C). However, the communications and computations with a higher pipeline degree will create higher overheads, which may counteract the benefit of pipelining and make the overall

*Corresponding author.

TABLE I: Notations

Name	Description
P	The number of workers (or GPUs) in the cluster.
α_{gemm}	Startup time of a GEMM operation on a GPU.
β_{gemm}	Calculation time per element of GEMM.
α_a	Startup time of all-to-all collective communication.
β_a	Transmission time per byte of all-to-all.
B	# of samples per GPU (or local mini-batch size).
L	# of tokens per sample (or sequence length).
N	# of assigned tokens per expert.
E	Total number of experts.
M	Embedding size of a token.
H	Hidden size of the feed-forward layer in experts.
k	Top-k experts should be selected for each token.

iteration time even longer. In addition, different MoE layers in a model may have different configurations, so their pipeline degrees may not be identical to achieve the best performance. Existing MoE training systems (FasterMoE [9] and Tutel [10]) only support manual configuration of fixed pipeline degrees for MoE layers, which is usually sub-optimal. To this end, we design an adaptive strategy to find the best pipeline degree for any given configurations of an MoE layer according to their computation and communication workloads, such that the training time is minimized. To achieve this, we propose three novel techniques: 1) the formulation of an optimization problem aiming to minimize the iteration time (§III), 2) robust performance models for predicting the computation and communication time for any workloads of MoE (§IV-A), and 3) an efficient and optimal solution with polynomial time complexity (§IV-B). We implement our PipeMoE atop Tutel and conduct extensive experiments on a 64-GPU cluster connected with 100Gb/s InfiniBand. Experimental results show that our PipeMoE almost always achieves better training performance than state-of-the-art MoE systems including Tutel [10] and FasterMoE [9]. Specifically, in all 522 tested MoE layers, our PipeMoE is almost always better than Tutel with manually set pipeline degrees and achieves an average of 8.3% improvement over Tutel. In real-world popular NLP models with MoE, PipeMoE runs 12%-29% and 5%-77% faster than Tutel and FasterMoE, respectively.

II. Background and Preliminaries

In this section, we introduce some background and preliminaries about MoE and distributed training. For ease of presentation, we summarize some frequently used notations throughout the paper in Table I.

A. MoE layer

In training sparsely activated large models, an MoE layer is normally used to replace the original feed-forward layer [6]. The MoE layer contains two main components: a gating function and E experts, as shown in Fig. 1. The input of the MoE layer is the output of its previous hidden layer whose shape is (B, L, M), where B is the number of samples per mini-batch (i.e., mini-batch size), L is the sequence length per sample, and M is the length of embedding of each token.

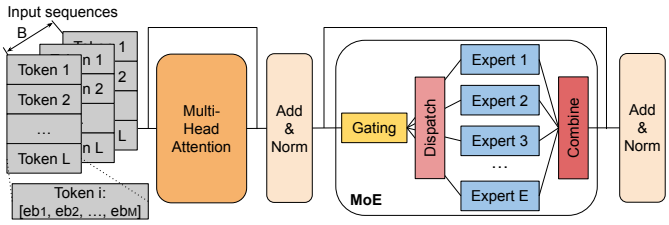


Fig. 1: A typical structure of MoE.

We use I to denote the input tensor of the MoE layer, then $I \in \mathbb{R}^{B\times L\times M}$.

Gating function. The input tensor I should be fed into a gating function g, which is composed of a small feed-forward neural network (typically with one to two layers) and a routing function (with a softmax layer and top-k selection) to select top-k experts for each input token, that is $G = g(I)$, which has a shape of $(E \times N \times M)$, i.e., $G \in \mathbb{R}^{E\times N\times M}$, where N is the maximum number of tokens assigned to all experts. The i^{th} row of G ($G[i,:,:]$) represents the data that should be *dispatched* to expert i ($i = 1, 2, ..., E$) for computation. Note that the gating function also generates the weight for each expert, we ignore this computation for ease of presentation as it does not affect our problem formulation and solution.

Experts. In the current mini-batch of training, expert i only processes the data of $G[i,:,:]$. All experts have an identical structure that can be executed in parallel with different input tokens. In general, the expert is also a small neural network with two identical workload feed-forward layers (the first layer has a size of $M \times H$ and the second layer is $H \times M$), where H is the size of the hidden layer. Let e_i denote expert i. The output of expert i can be represented by $Q_i = e_i(G[i,:,:])$. The outputs of different experts are then *combined* to generate the MoE output (i.e., $[Q_1, Q_2, ..., Q_E]$), which can be reshaped as $(B \times N \times M)$.

B. Data parallel and expert parallel

Data parallel. In distributed DL, the data-parallel training technique has become a de-facto method when the device memory is enough to hold the whole model parameters [17]–[19]. In data-parallel synchronous stochastic gradient descent (SGD), a mini-batch of $B \times P$ samples (X_t) is distributed to P workers on a P-worker cluster at iteration t, which means each worker still processes B samples (X_t^i) for the current mini-batch with model parameters W_t. It is also known as weak scaling. Before updating the model parameters, all workers should be synchronized to aggregate their local gradients $\nabla\mathcal{L}(W_t, X_t^i)$. That is

$$W_{t+1} = W_t - \eta\frac{1}{P}\sum_{i=1}^{P}\nabla\mathcal{L}(W_t, X_t^i). \tag{1}$$

It means that only the model parameters and aggregated gradients are identical among all GPUs, but numerical values of feed-forward and backpropagation computations are different as they are handling different data.

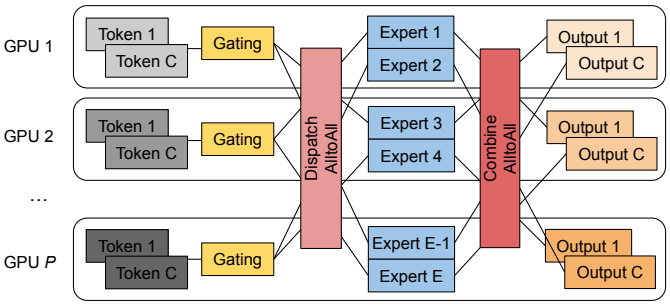


Fig. 2: An example of expert parallelism. Each GPU holds two experts. Tokens should be dispatched (and combined) to (and from) particular experts though all-to-all operations.

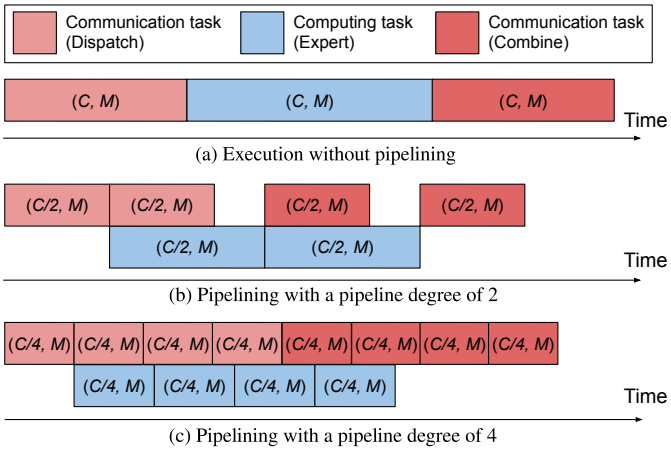


Fig. 3: Execution time of different pipeline degrees.

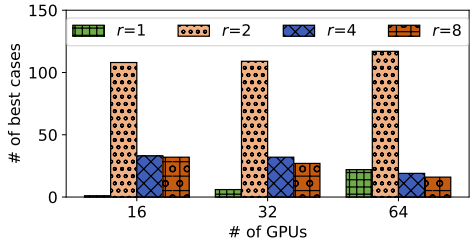


Fig. 4: The number of cases (the detailed configurations can be found in Table IV) with the best pipeline degree (r) in a search range of $\{1, 2, 4, 8\}$. While a pipeline degree of 2 occupies most cases that achieve the best training performance, there are about one third of the cases that require a different pipeline degree to achieve the best performance. The highest improvement of the best pipeline degree from $\{1, 2, 4, 8\}$ is 78%-102% over the worst pipeline degree.

Expert parallel. As shown in Fig. 2, when training MoE models on a distributed cluster, multiple experts on a single MoE layer will be placed on different workers as one worker has not enough memory to store all experts, which is called expert parallelism. Along with data parallel, each worker needs to calculate its own part of data (i.e., X_t^i for worker i). However, due to the dynamic gating in MoE, X_t^i may be routed to the experts that are not located in the same worker. Assume that there are E experts on an MoE layer running on P GPUs with expert parallelism, i.e., each GPU holds $\frac{E}{P} \geq 1$ experts. Let C denote the number of input tokens on each GPU, i.e., $C = B \times L$. Note that the input data for different GPUs is different in each iteration. As each token should be routed to k experts using top-k selection in the gating function, the tensor shape located in one GPU before dispatching can be denoted as (E, C_i, M), where $C_i = \frac{C \times k}{E}$. The data dispatch to different experts located on different GPUs is an all-to-all operation, which requires extensive communication costs. After that, the assigned tokens are fed to the feed-forward layers (*fflayers*) in each expert for computation. Finally, the outputs of all experts are combined to generate the output for the next layer, which can also be implemented with an all-to-all operation. The time-consuming parts of an MoE layer in expert parallelism are the two all-to-all operations (communication tasks) and the expert calculation (the computing task) as shown Fig. 3(a).

C. Pipelining computations and communications

The training time of an MoE layer mainly consists of computations of *fflayers* in experts (t_{expert}) and communications of all-to-all operations ($t_{dispatch}$ and $t_{combine}$). Thus, the MoE time of naive implementation (without pipelining as shown in Fig. 3(a)) can be represented by

$$t_{moe}^{naive} = t_{dispatch} + t_{expert} + t_{combine}. \quad (2)$$

To reduce the training time, the computing tasks in experts can be pipelined with the communication tasks of all-to-all operations by partitioning input data into multiple independent chunks [9,10].

In general, we assume that the input I_i of MoE on GPU i can be partitioned to d parts, $I_{i,1}, I_{i,2}, ..., I_{i,d}$, which are communicated (*dispatch*) through all-to-all sequentially. After $I_{i,j}$ has been dispatched, say $G_{i,j}$ at GPU i. $G_{i,j}$ can be immediately executed by the experts, i.e., $Q_{i,j} = e_k(G_{i,j})$. During this period, the all-to-all operation of $I_{i,j+1}$ can be executed simultaneously. It is similar to the execution of the *combine* operations. An example of MoE with a pipeline degree of 2 for pipelining is shown Fig. 3(b). An input tensor with shape (C, M) is partitioned to two equal-size smaller tensors, each of which has a shape of $(C/2, M)$. The first tensor is dispatched through all-to-all, which mainly occupies the network resources. The second tensor is also dispatched after the first one, during which the first tensor has been ready to be computed immediately in the expert layer, which mainly occupies GPU resources. After that, the output of the expert can be combined through all-to-all, so that the communication of combine can also be overlapped with the next expert computation. It is seen that Fig. 3(b) with a pipeline degree of 2 runs faster than Fig. 3(a) without pipelining.

Theoretically, the more fine-grained partition (i.e., higher pipeline degree) of the original tensor provides higher opportunities to overlap the execution of tasks. However, every operation (both computation and communication) in an MoE

layer has a constant startup time which is not related to the message size. Partitioning one tensor to two tensors to be executed sequentially takes a longer time to complete than executing once, such as computation [20] and communication [21]. As shown in Fig. 3(a)(b), two dispatch operations on shape $(C/2, M)$ separately take longer time than one dispatch operation on shape (C, M). An example of pipelining with a pipeline degree of 4 is shown Fig. 3(c), which indicates a higher degree of pipelining can ensure no idle resources during the training process, but it does not achieve shorter training time than Fig. 3(b). In our conducted experiments (see § V for detailed configurations) with 174 cases as shown in Fig. 4, not a single pipeline degree can always achieve the best performance. Even worse, a manually chosen pipeline degree can easily achieve the worst performance, which is 78%-102% worse performance than the best pipeline degree chosen from {1, 2, 4, 8} according to the results shown in Fig. 4. Thus, it is important to determine an optimal pipeline degree such that the training time is minimal. In this paper, our main goal is to find an optimal pipeline degree r to achieve minimal training time without taking extra costs.

III. Problem Formulation

The time consumption in the MoE layer can be categorized into two types: general matrix-to-matrix (GEMM) computation and all-to-all communication. We use $t_{gemm}(x, F)$ to denote a time function of computing GEMM for any two matrices $A \in \mathbb{R}^{m\times k}$ and $A \in \mathbb{R}^{k\times n}$ on a GPU with peak floating point performance of F, where $x = m \times k \times n$. For the all-to-all communication, we use $t_{a2a}(y, \alpha, \beta, P)$ to denote the all-to-all communication time for a tensor with y elements on a P-GPU cluster, where α and β are the startup time and the transmission time per element for communicating a message between two GPUs. On a particular P-GPU cluster, F, α, β, and P are fixed, so we use $t_{gemm}(x)$ and $t_{a2a}(y)$ to represent $t_{gemm}(x, G)$ and $t_{a2a}(y, \alpha, \beta, P)$ respectively for ease of presentation.

Assume that the hidden dimension of the expert in an MoE layer is H. The computation of expert is composed of two GEMM computations whose input dimensions are $(N \times M)$ vs. $(M \times H)$ and $(N \times H)$ vs. $(H \times M)$. The total workloads of these two GEMMs are the same, i.e., $x = N \times M \times H$. Thus, the expert computation time can be represented by

$$t_e = 2t_{gemm}(N \times M \times H). \tag{3}$$

The input dimension for the *dispatch* and *combine* operations is $N \times M$. Thus the communication time of dispatch (t_d) and combine (t_c) using all-to-all can be represented by

$$t_d = t_c = t_{a2a}(N \times M). \tag{4}$$

To reduce the training time, we would like to enable the computing tasks (i.e., expert computation) and the communication tasks (i.e., dispatch and combine) to be executed in parallel so that some communication time can be hidden.

Without loss of generality, we assume that the input data I for dispatch is partitioned to r parts along with the token dimension so that each part has N/r tokens. That is

$$I[:, :, :] \longrightarrow [I_1, I_2, ..., I_r], \tag{5}$$

where $I_j = I[:, (j-1)N/r : jN/r - 1, :]$ for $j = 1, 2, ..., r$. The dimension of I_j is r times smaller than I, so the workload for expert computation and all-to-all communications on each part is r times smaller than the non-partitioning version, but the overall workload remains unchanged. Using the r-degree partition, there are r times sequential *dispatch* communications, r times sequential expert computations, and r times sequential *combine* communications. Each part of the data is independent with each other, so the operations on different parts can be pipelined. We use the following three notations to denote the operations of *dispatch* communications, expert computations, and *combine* communications respectively.

$$\mathbb{D} = [D_1, D_2, ..., D_r], \tag{6}$$

$$\mathbb{E} = [E_1, E_2, ..., E_r], \tag{7}$$

$$\mathbb{C} = [C_1, C_2, ..., C_r]. \tag{8}$$

Let $t_d^{(i)}$, $t_e^{(i)}$, and $t_c^{(i)}$ to denote the elapsed-time of dispatch communication on D_i, expert computation on E_i, and combine communication on C_i, respectively. Let $\tau_d^{(i)}$, $\tau_e^{(i)}$, and $\tau_c^{(i)}$ denote the beginning timestamp of dispatch communication on D_i, expert computation on E_i, and combine communication on C_i, respectively. Note that $\mathbb{D}$ and $\mathbb{C}$ are communication tasks which mainly consume network resources, and $\mathbb{E}$ are computation tasks which consume GPU resources. To avoid the resource contention among the same types of tasks, we assume that any two tasks cannot occupy the same hardware resources. For example, while D_1 and D_2 can be executed simultaneously as they do not have data dependency, these two tasks should be executed sequentially as they both need to occupy the network resources. According to the assumption, we can derive the relationship among $\tau_d^{(i)}$, $\tau_e^{(i)}$, and $\tau_c^{(i)}$ as follows.

$$\tau_d^{(i)} = \begin{cases} 0, & i = 1, \\ \tau_d^{(i-1)} + t_d^{(i-1)}, & 2 \le i \le r. \end{cases} \tag{9}$$

$$\tau_e^{(i)} = \begin{cases} \tau_d^{(1)} + t_d^{(1)}, & i = 1, \\ \max\{\tau_e^{(i-1)} + t_e^{(i-1)}, \tau_d^{(i)} + t_d^{(i)}\}, & 2 \le i \le r. \end{cases} \tag{10}$$

$$\tau_c^{(i)} = \begin{cases} \tau_e^{(1)} + t_e^{(1)}, & i = 1, \\ \max\{\tau_e^{(i)} + t_e^{(i)}, \tau_c^{(i-1)} + t_c^{(i-1)}\}, & 2 \le i \le r. \end{cases} \tag{11}$$

Eq. 9 indicates that 1) D_1 can start immediately as we only consider training time of the MoE layer and 2) D_i starts after D_{i-1} has completed for $i \ge 2$. Eq. 10 indicates E_i starts after E_{i-1} and D_i have completed, and Eq. 11 indicates C_i starts after E_i and C_{i-1} have completed for $i \ge 2$. The overall time of the MoE layer can be represented by

$$t_{moe} = \begin{cases} t_d + t_e + t_c, & r = 1, \\ \tau_c^{(r)} + t_c^{(r)} - \tau_d^{(1)}, & r \ge 2. \end{cases} \tag{12}$$

The equation indicates that the MoE time depends on how we partition the tensors to be pipelined with different values of r. Our goal is to find the best r such that t_{moe} is minimal.

IV. Solution

To solve the above problem, we should be able to know the communication and computation times. Thus, we need to model the performance in both communication and computation such as we can predict their execution time with different sizes of input. In this section, we first build simple yet effective performance models for GEMM computation and all-to-all communication, then we derive the optimal solution to the problem of minimizing Eq. 12, followed by proposing an algorithm to be integrated into the existing distributed MoE training system.

A. Performance models

Though [9] has proposed a computation model for GEMM and a communication model for point-to-point data transmission, both models exploit simplified linear models without considering system overheads (such as the overhead in launching GPU kernels [20] and the CPU processing time when invoking a TCP/IP transmission between two nodes [21]). In this work, we propose more generic performance models for computation and communication respectively.

1) Performance model of computation: Partitioning large tensors to small ones means that GPU resources may be underutilized when the tensor size is small. Therefore, instead of modeling the GEMM computation as a zero-intercept linear equation [9], we add an intercept term to describe the overhead in launching GPU kernels as the following equation.

$$t_{gemm}(x) = \alpha_{gemm} + \beta_{gemm} \times x, \tag{13}$$

where x is the workload, i.e., $x = m \times n \times k$ for $A \times B$ with $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$. Plug Eq. 13 into Eq. 3, we have

$$t_e(n_e) = 2\alpha_{gemm} + 2\beta_{gemm} \times n_e, \tag{14}$$

where $n_e = N \times M \times H$. For ease-of-presentation, we use $\alpha_e = 2\alpha_{gemm}$ and $\beta_e = 2\beta_{gemm}$. Thus,

$$t_e(n_e) = \alpha_e + \beta_e \times n_e, \tag{15}$$

2) Performance model of communication: Similar to the performance of computation, we should also introduce startup time for the all-to-all operation. It is known that all-to-all algorithms are composed of multiple rounds of peer-to-peer communications [22,23], and each communication between two nodes can be described by the α-β model [21], where α is the startup time when communicating a message between two nodes and β is the transmission time per byte. Therefore, we can use the linear model with an intercept term to describe the elapsed time of an all-to-all operation, i.e.,

$$t_{a2a}(y) = \alpha_a + \beta_a \times y, \tag{16}$$

where y is the input dimension of data for the all-to-all operation, α_a and β_a are two terms related to the network speed between any two workers and the number of workers in the cluster. Some commonly used all-to-all algorithms have different α_a and β_a as shown in Table II[1]. Note that α and β are hardware-related parameters, which determine the communication time of transferring a message of n elements between two peers with $\alpha + \beta \times n$ [21]. Plug Eq. 16 into Eq. 4, we have

$$t_d(n_d) = \alpha_a + \beta_a \times n_d, \tag{17}$$

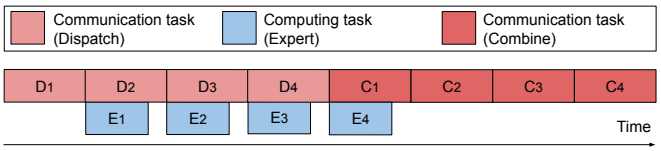
where $n_d = N \times M$. Similarly,

$$t_c(n_c) = \alpha_a + \beta_a \times n_c, \tag{18}$$

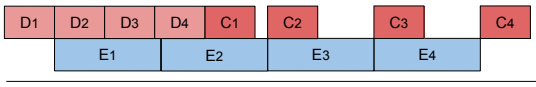
where $n_c = N \times M$.

TABLE II: Complexity of all-to-all algorithms (P workers).

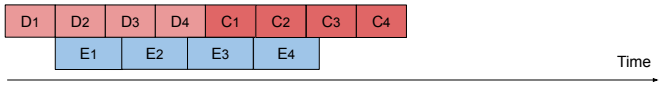
Algorithm	α_a	β_a
1-factor [25]	$(P-1) \times \alpha$	$(P-1) \times \beta$
Ring [26]	$(P-1) \times \alpha$	$\frac{P(P-1)}{2}\beta$



(a) **Q1** is true.



(b) **Q1** is false and **Q2** is true.



(c) **Q1** is false and **Q2** is false.

Fig. 5: Examples of $r = 4$ under three different conditions.

B. Optimal solution

According to the above performance models in both computation and communication, we can solve the optimal r to minimize Eq. 12. There are some max functions in Eq. 10 and Eq. 11. We need to decouple them with separate cases to eliminate the max functions for $r \geq 2$. In the tensor partition of MoE training, there are two good properties: 1) the partitioned tensors have the same shape, which means that they have the same communication time and the same computation time for all partitioned tensors, and 2) *dispatch* and *combine* operations have the same workload, which implies that their communication time is same.

For $r = 1$, we can directly predict the training time by

$$t_1^{moe} = 2t_d + t_e = 2\alpha_a + \alpha_e + 2\beta_a n_d + \beta_e n_e. \tag{19}$$

For $r \geq 2$, we need to decouple the max functions in Eq. 10 and Eq. 11. For a given r, we define a condition

$$t_d(\frac{n_d}{r}) \geq t_e(\frac{n_e}{r}), \tag{20}$$

[1]There exist some hierarchical algorithms [10,14,15,24] to better utilize the bandwidth of intra-node interconnect, which can also be modeled by Eq. 16

which is equivalent to

$$\textbf{Q1: } \alpha_a + \beta_a \frac{n_d}{r} \geq \alpha_e + \beta_e \frac{n_e}{r}. \tag{21}$$

We decouple the max functions with the following cases.

*1) **Q1** is true:* If **Q1** is true, it indicates that the communication time of the dispatch operation is larger than the computation time of the expert operation on a partitioned tensor. Thus, the expert computation time consumed by E_i can be fully hidden by the concurrent communication time consumed by D_{i+1}. In addition, in the overlapping between expert computation and *combine* communication, E_k can also be fully hidden by C_1. An example with $r = 4$ shown in Fig. 5(a) when **Q1** is true, which shows the four expert computing tasks are fully hidden by the communication tasks. So we have

$$\tau_e^{(i-1)} + t_e^{(i-1)} \leq \tau_d^{(i)} + t_d^{(i)}, \tag{22}$$

which can be used to eliminate the max functions in Eq. 10 and Eq. 11. We can obtain

$$\tau_e^{(i)} = \tau_d^{(i)} + t_d^{(i)}, \text{ for } i = 1, 2, ..., r. \tag{23}$$

$$\tau_c^{(i)} = \begin{cases} \tau_e^{(1)} + t_e^{(1)}, & i = 1; \\ \tau_c^{(i-1)} + t_c^{(i-1)}, & 2 \leq i \leq r. \end{cases} \tag{24}$$

It indicates that all r expert computations are fully hidden by the communications. Therefore, for $r \geq 2$,

$$\begin{aligned} t_2^{moe} = f_2(r) &= \tau_c^{(r)} + t_c^{(r)} - \tau_d^{(1)} \\ &= \sum_{i=1}^{r} t_d^{(i)} + \sum_{i=1}^{r} t_c^{(i)} = 2\sum_{i=1}^{r} t_d^{(i)} \\ &= 2r(\alpha_a + \beta_a \frac{n_d}{r}) \\ &= 2r\alpha_a + 2n_d\beta_a. \end{aligned} \tag{25}$$

Obviously, $f_2(r)$ is increased with larger r, which means r equals to 2 such that t_{moe} is minimal if **Q1** is true. Since

$$\textbf{Q1 is true } \implies \alpha_a + \beta_a \frac{n_d}{r} \geq \alpha_e + \beta_e \frac{n_e}{r} \tag{26}$$

$$\implies r \geq \frac{\beta_e n_e - \beta_a n_d}{\alpha_a - \alpha_e}, \tag{27}$$

we obtain

$$r = \max\{2, \lceil \frac{\beta_e n_e - \beta_a n_d}{\alpha_a - \alpha_e} \rceil\} \tag{28}$$

such that $f_2(r)$ is minimal, denoted as t_2^*.

*2) **Q1** is false and **Q2** is true:* If **Q1** is false, it indicates that the computation time of an expert operation is larger than the communication time of an all-to-all operation. Thus, either the communications are fully overlapped with computations except D_1 and C_r (e.g., Fig. 5(b)) or the computations are fully overlapped by the communications (e.g., Fig. 5(c)). We can distinguish these two cases with a condition

$$2\sum_{i=2}^{r-1} t_d^{(i)} < \sum_{i=1}^{r} t_e^{(i)}, \tag{29}$$

which means the overall communication time except D_1 and D_r is smaller than the overall computation time. It is equivalent to

$$\textbf{Q2: } 2(r-1)\alpha_a + \frac{2(r-1)}{r}\beta_a n_d < r\alpha_e + \beta_e n_e. \tag{30}$$

Thus, if **Q1** is false and **Q2** is true, Eq. 10 and Eq. 11 become

$$\tau_e = \begin{cases} \tau_d^{(1)} + t_d^{(1)}, & i = 1, \\ \tau_e^{(i-1)} + t_e^{(i-1)}, & 2 \leq i \leq r. \end{cases} \tag{31}$$

$$\tau_c^{(i)} = \tau_e^{(i)} + t_e^{(i)}, \text{ for } 1 \leq i \leq r. \tag{32}$$

We can obtain

$$\begin{aligned} t_3^{moe} &= \tau_c^{(r)} + t_c^{(r)} - \tau_d^{(1)} \\ &= t_d^{(1)} + \sum_{i=1}^{r} t_e^{(i)} + t_c^{(r)} \\ &= 2(\alpha_a + \beta_a \frac{n_d}{r}) + r(\alpha_e + \beta_e \frac{n_e}{r}) \\ &= 2\alpha_a + \beta_e n_e + \frac{2\beta_a n_d}{r} + \alpha_e r. \end{aligned} \tag{33}$$

Therefore, to find its minima, t_3^*, we should solve

$$\text{minimize: } f_3(r) = 2\alpha_a + \beta_e n_e + \frac{2\beta_a n_d}{r} + \alpha_e r \tag{34}$$

$$\text{s.t. } \alpha_a + \beta_a \frac{n_d}{r} < \alpha_e + \beta_e \frac{n_e}{r} \tag{35}$$

$$2(r-1)\alpha_a + \frac{2(r-1)}{r}\beta_a n_d < r\alpha_e + \beta_e n_e \tag{36}$$

$$r \geq 2, \tag{37}$$

which is a convex problem and can be solved efficiently.

*3) **Q1** is false and **Q2** is false:* If **Q1** is false and **Q2** is false, then

$$\tau_e^{(i)} = \tau_d^{(i)} + t_d^{(i)}, \text{ for } i = 1, 2, ..., r, \tag{38}$$

and

$$\tau_c^{(i)} = \begin{cases} \tau_e^{(1)} + t_e^{(1)}, & i = 1; \\ \tau_c^{(i-1)} + t_c^{(i-1)}, & 2 \leq i \leq r, \end{cases} \tag{39}$$

which is the same as Eq. 23 and Eq. 24 when **Q1** is true. The MoE time becomes

$$t_4^{moe} = 2r\alpha_a + 2n_d\beta_a. \tag{40}$$

To achieve its minimal time t_4^*, we should

$$\text{minimize: } f_4(r) = 2\alpha_a r + 2n_d\beta_a \tag{41}$$

$$\text{s.t. } \alpha_a + \beta_a \frac{n_d}{r} < \alpha_e + \beta_e \frac{n_e}{r}, \tag{42}$$

$$2(r-1)\alpha_a + \frac{2(r-1)}{r}\beta_a n_d \geq r\alpha_e + \beta_e n_e, \tag{43}$$

$$r \geq 2, \tag{44}$$

which is a linear programming problem.

To summarize, we derive the following theorem to determine the best r such that the training time of MoE is minimal.

Theorem 1. *Given an MoE layer running on a P-worker cluster with expert parallelism using the all-to-all collective*

for communication, we can partition the input tensor with a pipeline degree of r for pipelining. To achieve minimal training time, r should satisfy

$$r = \begin{cases} 1, & t_i^* \geq t_1 \text{ for } \forall i \in \{2,3,4\}, \\ \underset{r}{\text{argmin}} \ \min\{t_2^*, t_3^*, t_4^*\}, & t_i^* < t_1 \text{ for } \exists i \in \{2,3,4\}. \end{cases} \tag{45}$$

Proof. As discussed in §IV-B1, §IV-B2, and §IV-B3, they have covered all cases for $r \geq 2$. Therefore, if $\forall i \in \{2,3,4\}, t_i^* \geq t_1$, it means all $r \geq 2$ cannot achieve shorter time than $r = 1$. Otherwise, there exists a case such that $r \geq 2$ can achieve shorter time than $r = 1$ (i.e., $\exists i \in \{2,3,4\}, t_i^* < t_1$), so should choose the r such that $\min\{t_2^*, t_3^*, t_4^*\}$ is minimal. So we complete the proof.

□

Algorithm 1 FindOptimalPipelineDegree

Input: $B, L, E, H, M, P, k, \alpha_e, \beta_e, \alpha_a$, and β_d
Output: r

1: $l = E/P$; ▷ Local experts per GPU
2: $n_d = B \times L \times M \times l \times k$;
3: $n_e = B \times L \times M \times H \times l \times k$;
4: $r_1 = 1$;
5: $t_1 = 2 \times (\alpha_e + \beta_e \times n_e) + \alpha_a + \beta_a n_d$; ▷ Eq. 19
6: $r_2 = \max\{2, \lceil \frac{\beta_e n_e - \beta_a n_d}{\alpha_a - \alpha_e} \rceil\}$;
7: $t_2 = 2r_2\alpha_a + 2n_d\beta_a$;
8: $r_3, t_3 = \text{solve}(f_3)$; ▷ Solve with SLSQP
9: $r_4, t_4 = \text{solve}(f_4)$;
10: **if** $\min(t_2, t_3, t_4) \geq t_1$ **then**
11: $r = r_1$;
12: **else**
13: candidate_mins = [t_2, t_3, t_4];
14: candidates = [r_2, r_3, r_4];
15: r = candidates[argmin(candidate_mins)];
16: **end if**
17: return r;

C. Algorithm

For an MoE layer that has parameters B, L, E, H, M, k that are related to the model, and $P, \alpha_e, \beta_e, \alpha_a, \beta_a$ that are related to the cluster, we determine the optimal pipeline degree to partition the input data according to Algorithm 1. Note that $\alpha_e, \beta_e, \alpha_a, \beta_a$ can be estimated before training for a particular cluster (see §V-B). Lines 1-3 construct the dimensions for the performance models according to the MoE input. Lines 4-9 compute all the candidates with estimated time and their corresponding pipeline degrees according to t_1^{moe}, t_2^*, t_3^*, and t_4^*. Note that the "solve" function can be implemented with a sequential least squares programming (SLSQP) [27] solver. Lines 10-15 compute the best pipeline degree according to Theorem 1. The algorithm is quadratic convergence in solving f_3 and f_4 (Lines 8 and 9), and other operations take O(1) time complexity in minimizing Eq. 12. We plug our Algorithm 1 into the MoE module in the Tutel system[2] [10] which is a highly optimized MoE implementation and supports asynchronous execution of communication and computing tasks with different pipeline degrees. Tutel has also been used as a default MoE training module by Fairseq[3] [28]. Note that for any given MoE model, the input for Algorithm 1 can be known after the models have been built in the training script, so we can run the algorithm to configure the pipeline degree before training so that it would not affect the training performance.

[2]https://github.com/microsoft/tutel

TABLE III: The server configuration in our testbed.

Name	Model
CPU	Dual Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz
GPU	4x Nvidia RTX2080Ti (@1.35GHz and 11GB Memory)
Memory	512GB DDR4
PCIe	3.0 (x16)
Network	Mellanox MT27800 Family (ConnectX-5) @ 100Gb/s

V. Evaluation

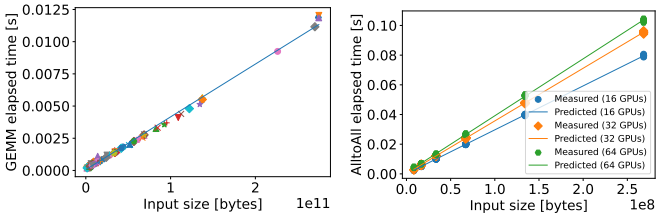
In this section, we first evaluate the accuracy of our performance models on computation and communication (i.e., Eq. 13 and Eq. 16) on a 64-GPU cluster with 100Gb/s InfiniBand. Then we compare the iteration time of our PipeMoE in training MoE layers with 522 configurations with Tutel on our testbed. After that, we demonstrate the performance in training real-world popular NLP models with MoE, followed by some discussions on the experimental results. Note that PipeMoE is a system-level optimization and has no side-effect in affecting the numerical results of convergence, so we mainly study its time performance.

A. Experimental settings

Testbed. We conduct all experiments on a 64-GPU cluster, which consists of 16 nodes connected with 100Gb/s InfiniBand. Each node is installed with four Nvidia Geforce RTX2080Ti GPUs connected with PCIe3.0x16. More details about the hardware of the server are shown in Table III. The distributed training system is Tutel atop PyTorch-1.10 [29] under a software environment of Ubuntu-18.04, CUDA-10.1, cuDNN-7.6, OpenMPI-4.0.1, and NCCL-2.12.

MoE configurations. To cover a variety of typical configurations of MoE layers, we choose a combination of input parameters whose ranges are shown in Table IV. Note that some cases that require memory larger than the capacity of GPU memory (11GB) cannot run on our cluster are excluded, so we conduct totally 522 valid cases for the experiments. In addition, similar to FasterMoE [9], we also choose two popular NLP models, i.e., Transformer [30] for the translation task on the iwslt2014 German-English translation dataset[4], and RoBERTa [31] for the language modeling task on the wikitext-103 dataset [32]. We replace the feed-forward layers in Transformer and RoBERTa with MoE layers to construct MoE models. For different numbers of GPUs, we configure different numbers of experts in MoE layers to ensure each GPU contains one expert or two experts as shown in Table V.

[3]https://github.com/facebookresearch/fairseq
[4]https://workshop2014.iwslt.org/



(a) GEMM on RTX2080Ti. (b) All-to-all.

Fig. 6: Performance models of computation and communication. Markers are measured values and lines are predicted values with estimated parameters. (a) $\alpha_{gemm} = 6.19 \times 10^{-5}$ and $\beta_{gemm} = 4.1 \times 10^{-14}$. (b) (α_a, β_a) equals to $(1.72 \times 10^{-5}, 2.96 \times 10^{-10})$, $(2.09 \times 10^{-4}, 3.54 \times 10^{-10})$, and $(7.83 \times 10^{-4}, 3.84 \times 10^{-10})$ on 16-GPU, 32-GPU, and 64-GPU environments, respectively.

Due to the memory constraint in our testbed, we configure the number of local experts on each GPU to 1 or 2.

TABLE IV: Configurations of MoE layers. E/P indicates the number of local experts per GPU.

Parameter	Candidate Values
P	{16, 32, 64}
B	{2, 4, 8}
L	{512, 1024, 2048}
H	{1024, 2048, 4096, 8192}
M	{1024, 2048, 4096, 8192}
E/P	{1, 2}

TABLE V: Configurations of NLP models with MoE.

Model	Base (# of Params)	Dataset	MoE Configuration				
			B	L	H	M	E/P
TM-1	Transformer (235M)	iwslt2014-de-en	4	256	1024	512	1
TM-2			4	256	1024	512	2
RM-1	RoBERTa (27M)	wikitext-103	2	64	3072	768	1
RM-2			2	64	3072	768	2

B. Performance models

In Algorithm 1, we require the input parameters that are related to the cluster for the performance models of computation and communication. We measure the elapsed time with a range of sizes for the GEMM computation and the all-to-all communication to fit the performance models in Eq. 13 and Eq. 16. The results are shown in Fig. 6. It is seen that our linear models with intercept terms (i.e., startup time) can well fit the real measured performance. In all-to-all, α_a and β_a are different on different sizes of clusters as they are related to the number of GPUs as shown in Table II. For any new GPU cluster, we only need to estimate the parameters once by running micro-benchmarks with a range of input data before training models, which would not affect the training performance.

TABLE VI: Performance improvement of PipeMoE over Tutel on average. The numbers in the brackets are the highest improvements among the tested cases.

# of GPUs	Pipeline Degree			
	1	2	4	8
16	16.8% (43.1%)	11.5% (26.1%)	3.4% (26.0%)	8.6% (35.0%)
32	15.6% (33.8%)	3.5% (9.8%)	6.1% (33.9%)	12.9% (44.4%)
64	12.5% (34.9%)	5.3% (12.0%)	8.6% (42.5%)	17.2% (45.8%)

C. Training time on MoE layers

On the runnable 522 MoE layers from Table IV, we run our PipeMoE to measure the step time and compare it with the manually set pipeline degrees of {1, 2, 4, 8} as configured by Tutel [10] on clusters with different number of GPUs. The performance comparison is shown in Table VI. It is seen that PipeMoE with an adaptive pipeline degree achieves 3.4%-16.8%, 3.5%-15.6%, and 5.3%-17.2% performance improvement on average among the configured MoE layers over Tutel on 16-GPU, 32-GPU, and 64-GPU environments, respectively. As PipeMoE adaptively chooses a pipeline degree based on the MoE configuration, it can avoid the worst pipeline degree. In terms of the improvements over the worst pipeline degree chosen from {1, 2, 4, 8}, PipeMoE is 26.0%-43.1%, 9.8%-44.4%, and 12.0%-45.8% faster than Tutel on 16-GPU, 32-GPU, and 64-GPU environments, respectively. Among all the 522 cases, 86.1% cases have been solved to find better or equal pipeline degrees by PipeMoE than manually chosen from {1, 2, 4, 8}. Overall, our PipeMoE is around 8.3% faster than Tutel on average.

D. End-to-end time on real-world models

TABLE VII: Comparison of average iteration wall-clock time (mean±std) in seconds. We conduct 3 independent runs to take the average and each run trains 200 iterations to measure its average time. S_1 and S_2 are the speedups of PipeMoE over Tutel and FasterMoE respectively.

# of GPUs	Model	Time (s)			S_1	S_2
		Tutel	FasterMoE	PipeMoE		
16	TM-1	.335±.005	.314±.005	**.299**±.011	1.12×	1.05×
	TM-2	.418±.007	.383±.005	**.348**±.007	1.20×	1.10×
	RM-1	.657±.003	1.010±.001	**.580**±.002	1.13×	1.74×
	RM-2	.702±.004	1.026±.002	**.603**±.002	1.16×	1.70×
32	TM-1	.450±.011	.398±.032	**.370**±.012	1.22×	1.08×
	TM-2	.591±016	.506±.016	**.471**±.018	1.25×	1.07×
	RM-1	.679±.001	1.064±004	**.587**±.004	1.16×	1.81×
	RM-2	.742±.002	1.083±005	**.611**±.003	1.21×	1.77×
64	TM-1	.623±015	.515±.017	**.497**±.011	1.25×	1.04×
	TM-2	.880±006	.831±.016	**.684**±006	1.29×	1.21×
	RM-1	.800±.001	1.118±.005	**.665**±.001	1.20×	1.68×
	RM-2	.929±.001	1.158±.002	**.720**±.003	1.29×	1.61×

We measure the average iteration time of end-to-end training on the models listed on Table V and compare our PipeMoE with existing state-of-the-art MoE systems including Tutel [10] and FasterMoE [9]. We use the official implementations of these systems for experiments. The results are shown in Table VII. In most cases, Tutel runs faster than FasterMoE due to its dedicated optimization for MoE training. Our PipeMoE

is built atop Tutel and it achieves further improvement over Tutel and FasterMoE with adaptive pipelining. It can also be seen that on different sizes of clusters and models, PipeMoE always achieves better performance than Tutel and FasterMoE. Specifically, PipeMoE is 12%-29% and 5%-77% faster than Tutel and FasterMoE, respectively.

E. Discussions

In practice, the training process of large models contains other parallelisms between computing and communication tasks such as wait-free backpropagation (WFBP) [33], which is the default feature in existing distributed DL frameworks like PyTorch-DDP. In WFBP, the communication tasks of gradient aggregation (with all-reduce) can be overlapped with gradient computation of its previous layers [34]. This means that the all-to-all communications could be executed simultaneously with the all-reduce communications, so they may compete for network resources. Thus, the communication model in predicting the all-to-all time may not accurately reflect the real execution time. It would cause the chosen pipeline degree by PipeMoE runs slower than well chosen one. For example, in the case of $(B, L, M, H, E/P) = (2, 2048, 8192, 4096, 2)$ on the 64-GPU cluster, PipeMoE predicts a pipeline degree of 2, and the running MoE time is around 289ms. However, it can run faster with only 253ms with a pipeline degree of 8. It would be interesting to jointly schedule the communication tasks of all-to-all and all-reduce in MoE training. We leave this as our future work.

In the end-to-end training time, the overall improvement is also limited by Amdahl's law. First, the MoE layers occupy a ratio of R (e.g., $\sim 40\% - 60\%$) to the overall training time. Second, the pipelining technique in an MoE layer can only reduce the time that is smaller than the maximum of communication time and computation time. Therefore, any system-level optimization with task scheduling for MoE layers can only achieve a maximum of $50\% \times R$ improvement over the training system without pipelining. Thus, for an MoE model whose MoE time occupies not larger than 60% of the overall training time, PipeMoE can only achieve a maximum of 30% speedup over the non-pipelining version in the end-to-end training time. In future work, it could be worthy exploring data compression techniques to reduce the communication time of all-to-all and kernel optimizations to reduce the expert computation time to further improve the training performance.

VI. Related Work

There are three main orthogonal directions in optimizing the training performance of MoE models.

MoE algorithms. Starting from MoE techniques being applied in large-scale deep neural networks [5,6], there are different types of MoE structures to address the performance issues of unbalanced workloads. Lewis et al. [11] proposed BASE layers in MoE by formulating token-to-expert allocation as a linear assignment problem such that experts can receive similar workloads of token assignment. Zhou et al. [35] proposed to let experts choose tokens instead of tokens select experts to balance the workloads among different workers. There exist some optimizations improving the generalization capability of MoE models such as stochastic routing of experts [12] or adding dropout in gating [36].

Optimization of all-to-all. The training systems of MoE models highly rely on the all-to-all collective communication to dispatch data and combine results on distributed workers [6]. Some communication-efficient all-to-all algorithms were recently proposed to reduce communication time. For example, NCCL-v2.12 introduces a topology-aware all-to-all algorithm that leverages faster input and output ports to transfer data [13]. HetuMoE [14] proposed a hierarchical all-to-all algorithm that reduces the communication rounds in inter-node communications, which better utilizes high-bandwidth and low-latency intra-node interconnects like PCIe or NVLink. The 2D-hierarchical all-to-all algorithm [10,15,16,24] has a similar idea that tries to better utilize intra-node bandwidth.

Scheduling of MoE layers. Communication scheduling techniques have been widely used in accelerating distributed training [37]–[41] in both data parallelism and model parallelism. However, expert parallelism in MoE models is quite different from the existing pipelining techniques. Recent studies [9,10] try to identify the pipelining opportunities for communication tasks of all-to-all operations and computing tasks of $fflayers$. However, FasterMoE [9] only allows a pipeline degree of 2 to pipeline the expert computations and all-to-all communications. Tutel [10] enables a manually set degree of pipelining or a heuristic search under limited searching space, which may be sub-optimal.

VII. Conclusion

In this work, we present an adaptive pipelining algorithm named PipeMoE to train MoE models. PipeMoE automatically selects the best pipeline degree according to the configuration of MoE layers without extra computation overhead, so it also avoids the worst cases with manually chosen values. To achieve this, we proposed three novel techniques in PipeMoE: 1) generic performance models for computation and communication in training MoE layers, 2) a problem formulation on how to partition input data for achieving minimal training time, and 3) an optimal and efficient solution to the problem, which finds the best pipeline degree without introducing extra computing overhead. Extensive experiments on a 64-GPU cluster were conducted. The results showed that our PipeMoE outperforms state-of-the-art MoE training systems by 8.3% in average on 522 configured MoE cases. In real-world popular MoE models, PipeMoE runs 5%-77% faster in end-to-end training speed over state-of-the-art MoE training systems including Tutel and FasterMoE. The authors have provided public access to their code at https://github.com/Fragile-azalea/PipeMoE.

Acknowledgments

The research was supported in part by a RGC RIF grant under the contract R6021-20, and RGC GRF grants under the contracts 16209120 and 16200221.

REFERENCES

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[2] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," in *Proceedings of Machine Learning and Systems 2022*, 2022.

[3] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *Journal of Machine Learning Research*, vol. 23, no. 120, pp. 1–39, 2022.

[4] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991.

[5] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in *International Conference on Learning Representations*, 2017.

[6] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "GShard: Scaling giant models with conditional computation and automatic sharding," in *International Conference on Learning Representations*, 2021.

[7] C. Riquelme, J. Puigcerver, B. Mustafa, M. Neumann, R. Jenatton, A. Susano Pinto, D. Keysers, and N. Houlsby, "Scaling vision with sparse mixture of experts," *Advances in Neural Information Processing Systems*, vol. 34, pp. 8583–8595, 2021.

[8] Z. Ma, J. He, J. Qiu, H. Cao, Y. Wang, Z. Sun, L. Zheng, H. Wang, S. Tang, T. Zheng *et al.*, "BaGuaLu: targeting brain scale pretrained models with over 37 million cores," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2022, pp. 192–204.

[9] J. He, J. Zhai, T. Antunes, H. Wang, F. Luo, S. Shi, and Q. Li, "FasterMoE: modeling and optimizing training of large-scale dynamic pre-trained models," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2022, pp. 120–134.

[10] C. Hwang, W. Cui, Y. Xiong, Z. Yang, Z. Liu, H. Hu, Z. Wang, R. Salas, J. Jose, P. Ram *et al.*, "Tutel: Adaptive mixture-of-experts at scale," *arXiv preprint arXiv:2206.03382*, 2022.

[11] M. Lewis, S. Bhosale, T. Dettmers, N. Goyal, and L. Zettlemoyer, "BASE layers: Simplifying training of large, sparse models," in *International Conference on Machine Learning*. PMLR, 2021, pp. 6265–6274.

[12] S. Zuo, X. Liu, J. Jiao, Y. J. Kim, H. Hassan, R. Zhang, J. Gao, and T. Zhao, "Taming sparsely activated transformer with stochastic experts," in *International Conference on Learning Representations*, 2022.

[13] "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/, accessed: 2022-07-13.

[14] X. Nie, P. Zhao, X. Miao, and B. Cui, "HetuMoE: An efficient trillion-scale mixture-of-expert distributed training system," *arXiv preprint arXiv:2203.14685*, 2022.

[15] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation ai scale," *arXiv preprint arXiv:2201.05596*, 2022.

[16] R. Aminabadi, S. Rajbhandari, A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. Rasley, and Y. He, "Deepspeed-inference: Enabling efficient inference of transformer models at unprecedented scale," in *2022 SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2022, pp. 646–660.

[17] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[18] X. Jia, S. Song, S. Shi, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, and X. Chu, "Highly scalable deep learning training system with mixed-precision: Training ImageNet in four minutes," in *Proc. of Workshop on Systems for ML and Open Source Software, collocated with NeurIPS 2018*, 2018.

[19] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh, "Large batch optimization for deep learning: Training BERT in 76 minutes," in *International Conference on Learning Representations*, 2020.

[20] A. Li, B. Zheng, G. Pekhimenko, and F. Long, "Automatic horizontal fusion for GPU kernels," in *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2022, pp. 14–27.

[21] S. Sarvotham, R. Riedi, and R. Baraniuk, "Connection-level analysis and modeling of network traffic," in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, 2001, pp. 99–103.

[22] J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby, "Efficient algorithms for all-to-all communications in multiport message-passing systems," *IEEE Transactions on parallel and distributed systems*, vol. 8, no. 11, pp. 1143–1156, 1997.

[23] S. S. Vadhiyar, G. E. Fagg, and J. Dongarra, "Automatically tuned collective communications," in *SC'00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, 2000, pp. 3–3.

[24] L. Shen, Z. Wu, W. Gong, H. Hao, Y. Bai, H. Wu, X. Wu, H. Xiong, D. Yu, and Y. Ma, "SE-MoE: A scalable and efficient mixture-of-experts distributed training and inference system," *arXiv preprint arXiv:2205.10034*, 2022.

[25] S. E. Hambrusch, F. Hameed, and A. A. Khokhar, "Communication operations on coarse-grained mesh architectures," *Parallel Computing*, vol. 21, no. 5, pp. 731–751, 1995.

[26] A. Grama, V. Kumar, A. Gupta, and G. Karypis, *Introduction to parallel computing*. Pearson Education, 2003.

[27] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 1999. [Online]. Available: https://doi.org/10.1007/b98874

[28] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli, "fairseq: A fast, extensible toolkit for sequence modeling," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, 2019, pp. 48–53.

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[31] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[32] M. Stephen, X. Caiming, B. James, and R. Socher, "Pointer sentinel mixture models," in *International Conference on Learning Representations*, 2017.

[33] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing, "Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 181–193.

[34] A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," *arXiv preprint arXiv:1802.05799*, 2018.

[35] Y. Zhou, T. Lei, H. Liu, N. Du, Y. Huang, V. Zhao, A. Dai, Z. Chen, Q. Le, and J. Laudon, "Mixture-of-experts with expert choice routing," *arXiv preprint arXiv:2202.09368*, 2022.

[36] R. Liu, Y. J. Kim, A. Muzio, and H. Hassan, "Gating dropout: Communication-efficient regularization for sparsely activated transformers," in *International Conference on Learning Representations*. PMLR, 2022, pp. 13 782–13 792.

[37] S. Shi, X. Chu, and B. Li, "MG-WFBP: Efficient data communication for distributed synchronous SGD algorithms," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, 2019, pp. 172–180.

[38] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, and C. Guo, "A generic communication scheduler for distributed DNN training acceleration," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 16–29.

[39] Y. Bao, Y. Peng, Y. Chen, and C. Wu, "Preemptive all-reduce scheduling for expediting distributed DNN training," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 626–635.

[40] S. Shi, X. Chu, and B. Li, "Exploiting simultaneous communications to accelerate data parallel distributed deep learning," in *Proc. of INFOCOM*, 2021.

[41] Z. Luo, X. Yi, G. Long, S. Fan, C. Wu, J. Yang, and W. Lin, "Efficient pipeline planning for expedited distributed dnn training," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, 2022.