SCoMoE: Efficient Mixtures of Experts with Structured Communication

Anonymous authors
Paper under double-blind review

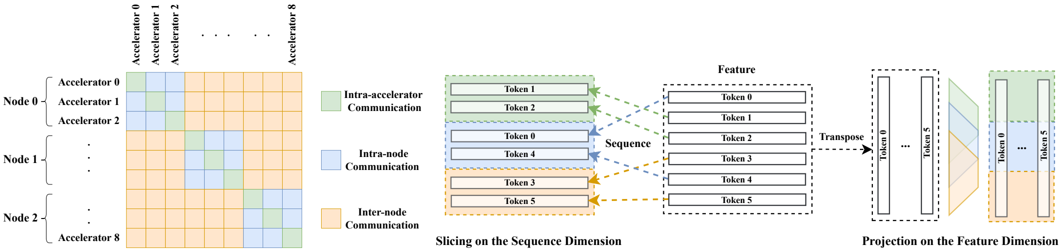
Abstract

Mixture-of-Experts (MoE) models are promising architectures for massive multilingual neural machine translation and large language models due to the advantage of sublinear scaling. However, the training of large MoE models is usually bottlenecked by the all-to-all communication (Lepikhin et al., 2020). To reduce the communication cost, we propose SCoMoE, an MoE architecture with structured all-to-all communication, inspired by the hierarchical architecture of the communication topology. SCoMoE encourages data to be communicated across devices through fast intra-accelerator/node communication channels, reducing communication throughput in the slow inter-node communication channel. We slice the data on the sequence dimension (SCoMoE-Seq) into three communication groups and project the data on the feature dimension (SCoMoE-Feat) into low-dimensional representations. To compensate the potential performance drop caused by the routing locality in SCoMoE, we further propose a token clustering approach to aggregating related tokens from different devices before the MoE layers. The sigmoid gating in the balanced router used in the token clustering is substituted with the softmax gating with differential sorting. Experiments on massive bilingual and multilingual machine translation demonstrate that our SCoMoE-Seq achieves a speedup of 1.5x over GShard with comparable performance, and SCoMoE-Feat outperforms Gshard significantly (1.2 BLEU) with a speedup of 1.32x.

1 Introduction

Recent years have witnessed a substantial interest in exploring sparse architectures based on Mixture of Experts for training massive multilingual machine translation (Lepikhin et al., 2020; Kim et al., 2021) and large language models (Fedus et al., 2021; Zhang et al., 2021; Ma et al., 2022; Du et al., 2021; Zoph et al., 2022; Rajbhandari et al., 2022; Lin et al., 2021). Experts of MoE models are distributed over multiple devices. Due to the sparse architecture where only a combination of experts are selected to process each input, the number of experts and hence the scale of MoE models can be sufficiently large while the computational cost is only sublinear to the number of parameters. Despite the advantage of efficient computation, MoE models require expensive all-to-all communication, to send the inputs and outputs of experts across the compute network. Previous study on GShard (Lepikhin et al., 2020) has shown that as MoE models scale, the all-to-all communication cost becomes the bottleneck for training.

To mitigate this issue, we propose Structured Communication based MoE (ScoMoE), which treats the all-to-all communication in a structured way rather than equally across different devices. The motivation behind SCoMoE is that the network bandwidth between devices and nodes is different across the compute network: the bandwidth inside an accelerator (intra-accelerator) is faster than that across accelerators, and the bandwidth inside a node (intra-node) is faster than that across nodes (inter-node). Figure 1a visualizes the hierarchical structure of communication topology with a 9×9 matrix, where different levels of communication are in different colors. We view the data flow in the all-to-all communication from the perspective of two dimensions: sequence dimension (tokens) and feature dimension (embeddings of tokens). The proposed SCoMoE transforms (slicing or projecting) the data flow at either the sequence or feature dimension into three communication groups: intra-accelerator, intra-node and global (inter-node) communication, as shown in Figure 1b. For the data slicing on the sequence dimension, we select tokens for a communication group according to



(a)

(b)

Figure 1: (a): The all-to-all communication contains three levels of communication with different bandwidth: the fast intra-accelerator communication inside each accelerator (green squares), the intra-node communication between Accelerator 0, 1 and Accelerator 3, 4 (blue squares), and the slow inter-node communication between Node 0 and Node 1 (orange squares). (b): Slicing data on the sequence dimension (left) and Projecting data on the feature dimension (right) into three groups corresponding to three levels of communication. Each row of the data is a token embedding.

the assignment scores of the tokens with the experts inside the group. To organize the data transformation on the feature dimension, we linearly project features into the three communication groups with lower feature dimensionality and recast them back after the all-to-all communication.

Theoretically, structuring all-to-all communication in this way is faster than its original form, since less data are transmitted through the slow inter-node communication channel. However, this may hurt the performance, because intra-accelerator/intra-node communication can be processed by only a part of experts. To alleviate this issue, we further propose a token clustering approach to aggregating related tokens from different devices, which elevates the association of tokens inside each device (accelerator/node). The proposed token clustering uses the balance router presented by Lewis et al. (2021) for clustering, where each device is a cluster. In the balanced router, a sigmoid gate is adopted to combine the inputs and outputs of experts, which could only broadcast the gradients to the activated experts, even though it may be more suitable to dispatch the tokens to other experts. Hence we propose to replace the sigmoid gate with the softmax gate via a straight-through trick (Bengio et al., 2013) for better gradient broadcasting.

In a nutshell, our contributions are summarized as follows:

1. We propose SCoMoE, that transforms the data flow in the all-to-all communication into three groups according to the bandwidth structure of communication topology.
2. A token clustering method is proposed to dispatch the related tokens to the same devices to alleviate the routing locality in the structured communication.
3. We propose to use softmax gate to substitue the sigmoid gate in the balanced router for better gradient broadcasting.
4. Experiments on massive bilingual and multilingual machine translation demonstrate that SCoMoE is faster than Gshard significantly with comparable or even better translation performance. Further analysis on training speed discloses the strategies of selecting hyper-parameters for SCoMoE.

2 Related Work

MoE models (Jacobs et al., 1991; Jordan & Jacobs, 1994) are ensemble method to integrate multiple experts. Shazeer et al. (2017) propose a gating network to select a combination of experts and mix data parallelism and model parallelism to increase the batch size. Gshard (Lepikhin et al., 2020) utilize the MoE parallelism proposed by Shazeer et al. (2017) to scale Transformer by replacing the MLP layer of Transformer with multiple MLP experts. Switch Transformer (Fedus et al., 2021) substitutes the top-2 routing of Gshard with the top-1 routing, and trains large sparse models with a selective precision strategy. Deepspeed-MoE (Kim et al., 2021) combines ZerO (Rajbhandari et al., 2020) and MoE parallelism to train large MoE models. Many recent large-scale pretrained language models (Fedus et al., 2021; Zhang et al., 2021; Ma et al., 2022; Du et al., 2021; Zoph et al.,

2022) have been trained with Gshard and its variants. A variety of new routers, e.g., hash router (Roller et al., 2021), random router (Zuo et al., 2021) and balanced router (Lewis et al., 2021), have been proposed in lieu of the top-2 router in GShard. We employ the balanced router proposed by the BASE Layer (Lewis et al., 2021) for token clustering. The balanced router uses the auction algorithm (Bertsekas, 1992) for balanced routing.

Although sparse MoEs are computationally efficient, they suffer from frequent and expensive all-to-all communication across the network. Gshard (Lepikhin et al., 2020) has shown that the all-to-all communication becomes the bottleneck for training large MoEs. To address this issue, Tutel (Rajbhandari et al., 2022) proposes a hierarchical all-to-all communication approach, which performs all-to-all communication and layout transformation inside a node, and then all-to-all communication across nodes. Although the hierarchical all-to-all communication of Tutel reduces the number of communication hops, it increases the amount of communication data. Hetu-MoE (Nie et al., 2022) also proposes a hierarchical all-to-all communication mechanism similar to that of Tutel. The difference is that Hetu-MoE dispatches all data inside one node to one accelerator, which may cause memory overflow. Sparse-MLP (Lou et al., 2022) shortens the communication time by eliminating 10% tokens before routing, according to the importance scores between tokens and experts. However, removing tokens may hurt the performance. Fast-MoE (He et al., 2021) optimizes the training speed of MoE on the computation of experts, rather than communication.

The proposed SCoMoE speeds up the all-to-all communication by dispatching data through the fast intra-accelerator/node communication. Both SCoMoE and previous hierarchical all-to-all communication algorithms (Rajbhandari et al., 2022; Nie et al., 2022), are proposed based on the hierarchical architecture of communication topology. Different from the hierarchical all-to-all communication of Tutel, which only works with a large number of nodes[1], SCoMoE could speed up MoEs significantly with a few nodes and a large batch size.

3 Model

3.1 Sparse Mixture-of-Experts and All-to-All Comminication

Sparse MoE models usually extend Transformer architecture with MoE layers where only a combination of experts are activated for computation each time. Different experts do not share parameters, as shown in Figure 2 (in blue), while other modules (in red) are shared across all devices.

Given an input sequence $\boldsymbol{X}$ which contains n tokens $\{\boldsymbol{X}_1, ..., \boldsymbol{X}_i, ..., \boldsymbol{X}_n\}$ with n token embeddings, sparse MoE models assign each token $\boldsymbol{X}_i$ to experts usually according to a token-to-expert assignment matrix $\boldsymbol{M}$:

$$\begin{aligned} \boldsymbol{M} &= \boldsymbol{X}\boldsymbol{W_E} \\ \boldsymbol{M}_{ij} &= \boldsymbol{W_{E_j}}^\top \cdot \boldsymbol{X}_i \end{aligned} \tag{1}$$

$\boldsymbol{W_E}$ is a projection matrix, where each column is corresponding to an expert embedding. $\boldsymbol{M}_{ij}$ is the assignment score between an expert embedding $\boldsymbol{W_{E_j}}^\top$ and a token embedding $\boldsymbol{X}_i$. Given the token-to-expert assignment matrix $\boldsymbol{M}$, the top-2 gating router selects the top-2 experts for each input token, while the balanced router assigns tokens to experts with the auction algorithm.

After the assignment, tokens are sent to their target experts via the all-to-all communication across devices, where each device sends/receives (i.e., dispatches and combines) tokens to/from all other devices. Experts in devices process the received tokens, and send the processed tokens back to their original devices, which requires another all-to-all communication. The all-to-all communication contains three levels of communication: intra-accelerator, intra-node, and inter-node, visualized in green, blue and orange respectively in Figure 1a. The intra-accelerator communication is actually copying data from the memory inside the corresponding accelerator, which is very fast. The intra-node communication is the communication across different accelerators inside one node, which is much faster than the inter-node communication across different nodes. However, most of the communication in the all-to-all communication are the slowest inter-node communication, which is visualized in orange in Figure 1a. Furthermore, the ratio of inter-node communication increases

[1] https://github.com/microsoft/tutel/blob/main/figure.svg

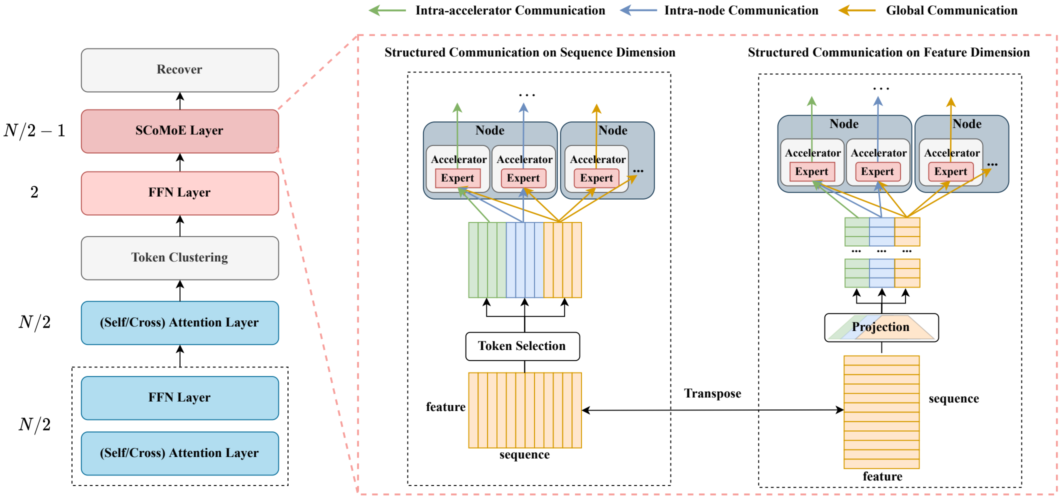


Figure 2: Architecture of the SCoMoE. The parameters of layers in blue are shared across different devices, while the parameters of layers in red are expert parameters. The token clustering aggregates related tokens at different devices together before structured communication layers. The aggregated tokens are sent back to the original devices (Recover) after the SCoMoE layers. The SCoMoE layers transform the data into three groups on either the sequence or feature dimension, and dispatch the transformed data to the experts via the intra-accelerator, intra-node and global communication respectively. We omit the process of combining the processed tokens into their original devices in the diagram for simplicity.

quadratically with the number of nodes. Therefore, the all-to-all communication becomes very expensive and hence the bottleneck for training large MoE models with many devices.

3.2 Structured All-to-All Communication

To reduce the all-to-all communication cost, we propose to transform the tokens for cross-device communication into three groups corresponding to the three levels of communication: intra-accelerator, intra-node and global communication.[2] The key idea is to increase the amount of tokens dispatched/combined in the fast intra-accelerator/node communication channels, and decrease the amount of tokens in the slow inter-node communication channel. Intuitively, this structured all-to-all communication acts like the multi-head attention where the inputs are projected into multiple keys, queries and values before attention computation in that our structured communication transforms the data into multiple groups for communication at different levels before expert computation.

Different from the vanilla MoE, we do not compute the token-to-expert assignment scores for all experts. Instead, we aim to select the experts inside each accelerator for the intra-accelerator communication, and the experts inside each node for the intra-node communication. As shown in Figure 2, the tokens in the intra-accelerator/node group can only be processed by the experts inside the accelerator/node, while the tokens in the global group are processed by all experts. The processed tokens are sent back to their original devices.

3.3 Data Transformation for Structured Communication

The data transformation for the structured communication could be either on the sequence or feature dimension. The difference of them is visualized in Figure 1b.

Data Slicing on the Sequence Dimension For the data transformation on the sequence dimension, we slice the tokens into each communication group according to the token-to-expert assignment scores introduced in the Section 3.1. The tokens to be sent to intra-accelerator/node devices have

[2] We do not perform data transformation for inter-node communication.

the highest assignment score with the experts inside the devices. Specifically, the tokens for local devices are collected via:

$$\mathbb{X}_{\text{local}} = \arg\operatorname*{TopK}_{i} \sum_{j \in \mathbb{E}_{\text{local}}} \boldsymbol{M}_{ij} \tag{2}$$

where $\boldsymbol{M}_{ij}$ is the assignment score between expert j and token i, $\mathbb{E}_{\text{local}}$ is a set of experts distributed at local devices, $\mathbb{X}_{\text{local}}$ is a set of tokens that have the top-k assignment scores with $\mathbb{E}_{\text{local}}$. k denotes the number of tokens to be dispatched to local devices, which is a hyper-parameter. The tokens for both the intra-accelerator and intra-node communication are collected in this way. We primarily select token for local communication, first intra-accelerator, then intra-node and finally for global communication. We mask the token-to-expert assignment scores of the selected tokens with $-\infty$ to prevent the same tokens from being selected for multiple groups of communication.

Data Projection on the Feature Dimension To transform the data on the feature dimension, we project the input sequence $\boldsymbol{X}$ with the dimensionality of d_{model} to lower-dimensional representations of $d_{\text{intra-accelerator}}$, $d_{\text{intra-node}}$ and d_{global}, with three different projections $\boldsymbol{W}_{\text{intra-accelerator}}$, $\boldsymbol{W}_{\text{intra-node}}$ and $\boldsymbol{W}_{\text{global}}$:

$$\begin{aligned}
\boldsymbol{X}_{\text{intra-accelerator}} &= \boldsymbol{X}\boldsymbol{W}_{\text{intra-accelerator}} \\
\boldsymbol{X}_{\text{intra-node}} &= \boldsymbol{X}\boldsymbol{W}_{\text{intra-node}} \\
\boldsymbol{X}_{\text{global}} &= \boldsymbol{X}\boldsymbol{W}_{\text{global}} \\
d_{\text{intra-accelerator}} + d_{\text{intra-node}} + d_{\text{global}} &= d_{\text{model}}
\end{aligned} \tag{3}$$

where $\boldsymbol{X} \in \mathbb{R}^{n\times d_{\text{model}}}$, $\boldsymbol{X}_{\text{intra-accelerator}} \in \mathbb{R}^{n\times d_{\text{intra-accelerator}}}$, $\boldsymbol{X}_{\text{intra-node}} \in \mathbb{R}^{n\times n_{\text{intra-node}}}$, $\boldsymbol{X}_{\text{global}} \in \mathbb{R}^{n\times d_{\text{global}}}$. $\boldsymbol{X}_{\text{intra-accelerator}}$, $\boldsymbol{X}_{\text{intra-node}}$, $\boldsymbol{X}_{\text{global}}$ are projected sequences to be dispatched via different communication channels. We display these projections and projected sequences in different colors in Figure 2.

The dimensionality of data dispatched to experts is reduced into $d_{\text{intra-accelerator}}$, $d_{\text{intra-node}}$ or d_{global}. Hence we also need to reduce the dimensionality of weights in experts correspondingly. As an expert is actually a fully-connected network, which contains two linear projections $\boldsymbol{W}_1 \in \mathbb{R}^{d_{\text{model}}\times d_{\text{ffn}}}$, $\boldsymbol{W}_2 \in \mathbb{R}^{d_{\text{ffn}}\times d_{\text{model}}}$.[3] Specifically, we set the shape of the first projection to be $(\hat{d}, d_{\text{ffn}})$, and that of the second projection to be $(d_{\text{ffn}}, \hat{d})$. $\hat{d} \in \{d_{\text{intra-accelerator}}, d_{\text{intra-node}}, d_{\text{global}}\}$, corresponding to the input dimension. Actually, the parameters of the experts are projected into three parts, each of which is corresponding to one level of communication. The outputs of experts are recast back to the original dimension (d_{model}) with another three projections.

3.4 Token Clustering

The data slicing on the sequence dimension forces more tokens to be dispatched to local devices, which may hurt the performance, as tokens dispatched to local devices are processed by only a few experts inside the devices. To mitigate this issue, we propose to apply token clustering before the SCoMoE layers. The token clustering is to aggregate related tokens to the same device (accelerator/node), where each device is a cluster. Aggregating related tokens together will make it better for the experts to process them. But the router of MoE usually assigns related tokens to the experts (Zoph et al., 2022), why do we need the token clustering? The reason is that the router of SCoMoE assigns more tokens to the experts inside local devices, while the token clustering could enhance the association of these tokens. We perform the token clustering before the first SCoMoE layer, and recover the original token distribution after the last SCoMoE layer.

We average the embeddings of experts inside an accelerator/node as the embedding of the device (either the accelerator or node), and assign tokens to devices in the same way as we assign tokens to experts. But unfortunately, the commonly used top-2 gating (Lepikhin et al., 2020) and top-1 gating (Fedus et al., 2021) routers could not be used here, since these routers abandon some tokens during routing. Therefore we resort to routers that keep all tokens, e.g, greedy router (Lewis et al., 2021) and balanced router (Lewis et al., 2021). The greedy router allows tokens to be dispatched to the top-1 expert according to the token-to-expert assignment matrix. However, different from the commonly used top-1 gating (Fedus et al., 2021), the greedy router does not set capacity for each

[3] We ignore the bias for simplicity.

expert, allowing any number of tokens to be sent to the same expert. When the greedy router is used to route tokens to devices, it may send too many tokens to a single device as it is unbalanced, causing out-of-memory issue. Incorporating the load balance loss (Lepikhin et al., 2020) does not help in this case, since the out-of-memory issue may immediately happen at the beginning of training. In contrast, the balanced router does not have the problem of the greedy router, which uses the auction algorithm (Bertsekas, 1992) to assign the same number of tokens to each expert. But the auction algorithm usually converges until many iterations, which is inefficient for autoregressive decoding. Hence the BASE Layer (Lewis et al., 2021) uses the balanced router instead of the greedy router at inference. In this work, we perform token clustering with the balanced router, and provide two options for inference: balanced or greedy routing.

The token clustering aggregates the tokens of different sentences together, which cannot be input to the self-attention layer. Therefore we have to change the order of Transformer layers, putting the attention layers before the MoE layers. The new architecture is shown in Figure 2, where the shared encoder/decoder layers are placed at the bottom, followed by the attention layers and MoE layers. The design of the new architecture is due to the following reasons: (1) Press et al. (2020) have investigated the impact of the order of attention layers and feed-forward layers on the performance and found that putting the attention layers before the feed-forward layers often yields good performance. (2) Riquelme et al. (2021) find that the top MoE layers are more important than the bottom MoE layers. (3) Intuitively, the training of sparse MoE layers is more unstable than the training of the dense layers (Zoph et al., 2022). Hence, putting the stable dense layers at the bottom may help stabilize the training of the entire architecture.

We put the balanced router before the first SCoMoE layer for token clustering. However, the token clustering also requires the all-to-all communication. We remove one MoE layer to avoid extra communication cost. Hence the number of SCoMoE layers is $N/2-1$ rather than $N/2$, as shown in Figure 2. To keep the same computation cost with the original model, we also insert two fully-connected layers after the balanced router.[4] The balanced router together with the two fully-connected layers act as the first MoE layer. Since the token clustering works on the sequence dimension, we only apply the token clustering to the SCoMoE layer that slices the data on the sequence dimension.

Differentiable Sorting The auction algorithm in the balanced router is not differentiable, hence the BASE Layer (Lewis et al., 2021) learns the expert embedding via the sigmoid gate (Srivastava et al., 2015):

$$\boldsymbol{X}_i' = \sigma(\boldsymbol{M}_{ij}) \cdot \text{expert}_j(\boldsymbol{X}_i) + (1 - \sigma(\boldsymbol{M}_{ij})) \cdot \boldsymbol{X}_i \tag{4}$$

where the output of the MoE layer $\boldsymbol{X}_i'$ is the combination of the expert input $\boldsymbol{X}_i$ and expert output $\text{expert}_j(\boldsymbol{X}_i)$ that are integrated with the normalized assignment score $\boldsymbol{M}_{ij}$. The gradients are broadcast through the sigmoid gate to update the parameters of the expert embeddings. But only the embeddings of the activated experts can receive the gradients, even though we have computed the assignment scores between any tokens and experts. With the gradients from the sigmoid gate, we only know how to update the assignment scores of the activated experts, which is not available for other experts. A straightforward solution to this is to replace the sigmoid gate with the softmax gate. However, the softmax operator requires multiple inputs, but only one expert is activated for each token. We notice that the auction algorithm assigns tokens to experts via sorting, which could be implemented as a linear projection with a sparse matrix, where each row and column only contains one non-zero value. Each row of the matrix could be seen as selecting a token from the sequence for an expert. We implement the sorting via linear projection:

$$\boldsymbol{X}_{\text{sort}} = \boldsymbol{W}_{\text{sort}} \cdot \boldsymbol{X} \tag{5}$$

where the sorted sequence $\boldsymbol{X}_{\text{sort}}$ is obtained by the projection $\boldsymbol{W}_{\text{sort}}$ on the input sequence $\boldsymbol{X}$. If the number of sorted tokens is n, and the number of experts is m, n/m tokens will be assigned to each expert. We visualize the sorting with a 4×4 sparse projection on a sequence of length 4 in Figure 3. The tokens in the sorted sequence are assigned to two experts. The assigned tokens often have the max or top-k scores with the expert, therefore each row could be seen as the argmax or argtopk on the token-to-expert assignment matrix. The softmax is the approximation to the argmax, if we set the temperature to a small value. Therefore, we share the gradients of the projection matrix with the

[4] Two fully-connected layer rather than one, since we use the top-2 gating in SCoMoE layers, where each token is processed by two experts.

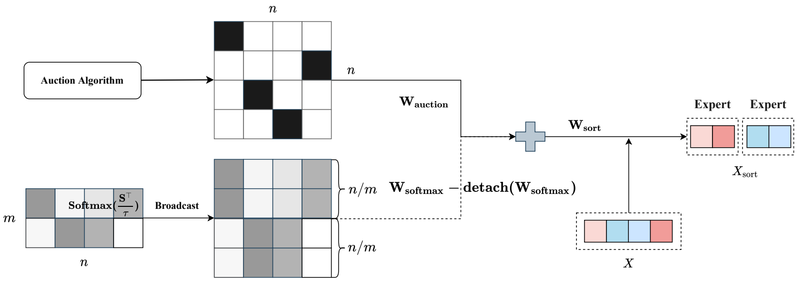

Auction Algorithm
n
n
$\mathbf{W}_{\text{auction}}$
$\mathbf{W}_{\text{sort}}$
Expert
Expert
X_{sort}
Softmax($\frac{\mathbf{S}^\top}{\tau}$)
Broadcast
m
n
n/m
n/m
$\mathbf{W}_{\text{softmax}} - \textbf{detach}(\mathbf{W}_{\text{softmax}})$
X

Figure 3: The visualization of differentiable sorting. The sorting projection $\boldsymbol{W}_{\text{sort}}$ from the auction algorithm, and the broadcast probability matrix $\boldsymbol{W}_{\text{softmax}}$ are combined together via the straight-through trick to sort the sequence. The sorted four tokens are divided into two groups, per corresponding to one expert.

softmax matrix via the straight-through trick (Bengio et al., 2013):

$$\begin{aligned} \boldsymbol{W}_{\text{sort}} &= \boldsymbol{W}_{\text{auction}} + \boldsymbol{W}_{\text{softmax}} - \text{detach}(\boldsymbol{W}_{\text{softmax}}) \\ \boldsymbol{W}_{\text{softmax}} &= \text{softmax}(\frac{\boldsymbol{S}^\top}{\tau}) \end{aligned} \tag{6}$$

$\boldsymbol{W}_{\text{sort}}$ is the combination of the sorting projection $\boldsymbol{W}_{\text{auction}}$ from the auction algorithm, $\boldsymbol{W}_{\text{softmax}}$, which is the normalized assignment matrix, and $\text{detach}(\boldsymbol{W}_{\text{softmax}})$, which has the same value as $\boldsymbol{W}_{\text{softmax}}$ but cannot broadcast the gradients. τ is the temperature which controls the sharpness of the softmax distribution. The lower the temperature is, the better the softmax approximates the argmax. $\boldsymbol{W}_{\text{sort}}$ is equal to $\boldsymbol{W}_{\text{auction}}$ in the forward computation. $\boldsymbol{W}_{\text{softmax}}$ has the same gradients as $\boldsymbol{W}_{\text{sort}}$ in the backward propagation, although $\boldsymbol{W}_{\text{softmax}}$ does not have influence on the training loss.

As the shape of the probability matrix $\boldsymbol{W}_{\text{softmax}}$ is different from that of $\boldsymbol{W}_{\text{sort}}$, (m, n) vs. (n, m), we cannot add them directly. Each row of $\boldsymbol{W}_{\text{softmax}}$ is a set of probabilities assigning tokens to an expert, which are corresponding to n/m rows of $\boldsymbol{W}_{\text{sort}}$ as we need to assign n/m tokens to each expert according to the same distribution. For the example in Figure 3, both the first and second row of sparse matrix $\boldsymbol{W}_{\text{auction}}$ are corresponding to the first row of the probability matrix $\boldsymbol{W}_{\text{softmax}}$. We reshape $\boldsymbol{W}_{\text{softmax}}$ from (m, n) to (n, n) by repeating each row of the matrix to n/m rows. In Figure 3, we reshape the matrix from (2,4) to (4,4), by repeating the first and second row of the matrix $\boldsymbol{W}_{\text{softmax}}$ to the first two and second two rows of the broadcast matrix.

4 Experiments

4.1 Experiment Settings

We evaluated the proposed SCoMoE on both bilingual and multilingual neural machine translation. In bilingual NMT experiments, we used the WMT17-En-Fr corpus, which consists of 35,762,532 training sentences, while in the multilingual NMT experiments, the OPUS-100 corpus (Zhang et al., 2020) was used, which consists of 100 languages and 107,924,846 training examples. The details of these two datasets are shown in Table 3 of Appendix A. We used the script of fairseq [5] to download and pre-process the WMT17-En-Fr dataset while the OPUS-100 dataset was directly downloaded from the OPUS web.[6] Both the WMT17-En-Fr and OPUS-100 dataset were tokenized via the BPE algorithm (Sennrich et al., 2016). For the multilingual experiments, a language token indicating the source language was prepended to each source sentence and similarly, a target language token was prepended to each target sentence.

Models were trained in two settings: base model and large model. The base models have 6 encoder/decoder layers, 8 experts and $d_{\text{model}} = 512$, which were trained on one node with 8 RTX A6000 GPUs. The large models have 12 encoder/decoder layers, 32 experts, and $d_{\text{model}} = 1024$,

[5] https://github.com/pytorch/fairseq/blob/moe/examples/translation/prepare-wmt14en2fr.sh

[6] https://opus.nlpl.eu/opus-100.php

Table 1: Translation performance of the base and large models on the test sets of OPUS-100 and WMT17-En-Fr. SCoMoE-(Seq/Feat)-γ_1: the SCoMoE-(Seq/MT) with the intra-accelerator and global communication.

Setting	Model	Multilingual				Bilingual	speedup
		All	En-Any	Any-En	win-rate (%)		
Base (0.2B)	Transformer	23.43	19.64	27.17	26.2	37.39	2x
	Gshard	24.41	20.02	**28.76**	-	37.96	1x
	Small-Capacity-Gshard	23.02	19.08	26.92	20.3	37.27	1.1x
	SCoMoE-Seq-γ_1	**24.45**	**20.67**	28.18	**43.3**	**38.24**	1.1x
	SCoMoE-Feat-γ_1	23.47	20.25	26.67	27.3	37.51	1.07x
Large (4.1B)	Gshard	26.78	23.11	**30.44**	-	**40.09**	1x
	Small-Capacity-Gshard	26.93	24.58	29.28	49.5	39.87	1.2x
	SCoMoE-Seq-γ_1	**27.19**	**25.68**	28.69	69.5	39.31	1.2x
	SCoMoE-Feat-γ_1	**27.99**	25.13	**30.85**	**73.9**	39.79	1.32x

which were trained on 4 nodes with 32 RTX A600 GPUs. The 4 nodes were connected with Infiniband, the bandwidth of which is 100 Gbps. The number of parameters in the base model is 0.2 billion while 4.1 billion in the large model. The details of the base and large model configuration could be found in Table 4 of Appendix A.

We mainly compared our method with Gshard and its variant with a small capacity factor on translation performance and training speed. We refer to our SCoMoE on the sequence dimension as SCoMoE-Seq, and that on the feature dimension as SCoMoE-Feat. The Gshard with a small capacity factor is referred to as Small-Capacity-Gshard. We set the ratio of tokens dispatched through the intra-accelerator communication to be γ_1 ($0 < \gamma_1 < 1$), and the ratio through the intra-node communication to be γ_2 ($0 < \gamma_2 < 1$), hence that through the global communication is $\gamma_3 = 1 - \gamma_1 - \gamma_2$ ($0 < \gamma_3 < 1$). To simplify the comparison, we set either γ_1 or γ_2 to be non-zero, and the other to be 0. We refer the SCoMoE-(Seq/Feat) with $\gamma_1 > 0, \gamma 2 = 0$ as SCoMoE-(Seq/Feat)-γ_1, while that with $\gamma_2 > 0, \gamma_1 = 0$ as SCoMoE-(Seq/MT)-γ_2. Please refer to Appendix A for more details about experiment settings.

4.2 Evaluation on Translation Performance

We evaluated the proposed methods on the WMT17-En-Fr and OPUS-100. We report the BLEU score on the WMT17-En-Fr, and the average BLEU scores over all, English-to-any, and any-to-English language pairs on the OPUS-100. Results are shown in Table 1. We also report the win-rate and the speedup of all models over Gshard.

We evaluated the training speed on the dummy-mt task [7] of fairseq, since the training speed evaluated on it is more stable. Each input sentence in the dummy-mt task is a series of numbers, e.g., "0, 1, 2, 3, 4, ...", rather than the real sentence. We fixed the sequence length of all source and target sentences to be 30, and trained models with 100 million dummy tokens without validation (It takes 920 steps to train each model).

Base Model The base SCoMoE was trained on a single node, hence we only need to consider the ratio for the intra-accelerator communication (γ_1). For a fair comparison on translation performance, we set γ_1 for SCoMoE-Seq to 0.3, and that for SCoMoE-Feat to 0.5, and the capacity factor of Small-Capacity-Gshard to 0.8, so that these models can be trained with a comparable training speed. More speed evaluations could be found in Appendix B.

From Table 1, we can observe that both Small-Capacity-Gshard and our SCoMoE-Seq are faster than Gshard by 10%. Although SCoMoE-Feat is slightly slower, it outperforms the Small-Capacity-Gshard on both the WMT17-En-Fr and OPUS-100 dataset. The SCoMoE-Seq is comparable or even better than Gshard on both the WMT17-En-Fr and OPUS-100 dataset, due to the proposed token clustering and differentiable sorting. We further demonstrate the effectiveness of the two methods

[7] https://github.com/pytorch/fairseq/blob/moe/fairseq/benchmark/dummy_mt.py

in Appendix C and D. Results of more settings on the communication are shown in Table 2, from which we observe significant speedups with a slight performance drop.

Large Model The large SCoMoE models were trained on multiple nodes. We set $\gamma_1 = 0.2$ for SCoMoE-Seq-γ_1, $\gamma_1 = 0.5$ for SCoMoE-Feat-γ_1, which can be trained in a training speed comparable to Small-Capacity-Gshard with a capacity factor of 1.0 (the default capacity factor is 1.25). From Table 1, we find that SCoMoE-Feat-γ_1 achieves the best performance in terms of both the training speed (1.32x) and the BLEU scores on the OPUS-100 dataset. It also performs comparably with Gshard on the WMT17-En-Fr dataset. The improvement of SCoMoE-Feat-γ_1 over Gshard may be due to the fact that SCoMoE-Feat-γ_1 projects each token embedding into different groups for different expert computations, i.e., computed by more experts.[8]

Table 2: Training speed and translation performance of SCoMoE-Seq with different settings on the test set of WMT17-En-Fr.

Setting	γ_1	γ_3	BLEU	Speedup
Base	0.3	0.7	**38.24**	1.1x
	0.5	0.5	38.13	1.17x
Large	0.2	0.8	39.31	1.2x
	0.5	0.5	**39.75**	**1.5x**

SCoMoE-Seq-γ_1 and Small-Capacity-Gshard have the same training speed, but the SCoMoE-Seq is better than Small-Capacity-Gshard on OPUS-100, in terms of both BLEU, and win-rate, while Small-Capacity-Gshard is better on WMT17-En-Fr. Why is Small-Capacity-Gshard worse than Gshard in the base setting, but comparable to Gshard in the large setting? The reason may be that we set the capacity factor of the base model to 1.0, and that of the large model to 1.25. It is not surprising that decreasing the capacity factor from 1.0 to 0.8 has a more significant negative impact on the performance than decreasing it from 1.25 to 1.0. We also observe that the SCoMoE-Seq is always better than Gshard on En-any translation, but worse on any-En translation, which may relate to the language specific routing inside the model.[9] We further increase the γ_1 of SCoMoE-Seq-γ_1 from 0.2 to 0.5 in Table 2 and observe a more significant speedup (from 1.2x to 1.5x) but no performance drop, even a slight improvement in terms of BLEU on WMT17-En-Fr. But unfortunately, SCoMoE-(Seq/Feat)-γ_2 are unable to converge due to the gradient overflow.

Other Experiments and Analysis

- We conduct the empirical evaluation and theoretical analysis on the training speed and communication time cost of SCoMoE in Appendix B.
- The effect of token clustering and differentiable sorting on the performance of SCoMoE can be found in Appendix C and D.
- The analysis on the sensitiveness of MoE models to the capacity factor at inference is provided in Appendix. E

5 Conclusion

In this paper, we have presented SCoMoE to reduce the all-to-all communication time of MoEs, by encouraging more data to dispatch through the fast intra-accelerator/node communication channel, and less data through the slow inter-node communication channel. By slicing/projecting the data on the sequence/feature dimension, we transform the data into three groups corresponding to three levels of communication. To alleviate the performance drop caused by routing locality, we further propose the token clustering to aggregate related tokens from different devices, and replace the sigmoid gate of the balanced router with the softmax gate through differentiable sorting. Experiments on massive bilingual and multilingual neural machine translation demonstrate that our SCoMoE speeds up Gshard significantly with comparable or even better performance. The empirical evaluation and theoretical analysis on the training speed and communication time cost discloses

[8] SCoMoE-Feat-γ_1 has the same amount of parameters and computation with Gshard, except for the three projections for data transformation.

[9] We leave the investigation on the routing behavior inside Gshard and SCoMoE to our future work.

the speedups achieved by our method under different setting and network bandwidth. The ablation study demonstrates the benefit of the proposed token clustering and differentiable sorting.

REFERENCES

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. URL http://arxiv.org/abs/1308.3432.

Dimitri P. Bertsekas. Auction algorithms for network flow problems: A tutorial introduction. *Comput. Optim. Appl.*, 1(1):7–66, 1992. doi: 10.1007/BF00247653. URL https://doi.org/10.1007/BF00247653.

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. URL https://arxiv.org/abs/2112.06905.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *CoRR*, abs/2101.03961, 2021. URL https://arxiv.org/abs/2101.03961.

Jiaao He, Jiezhong Qiu, Aohan Zeng, Zhilin Yang, Jidong Zhai, and Jie Tang. Fastmoe: A fast mixture-of-expert training system. *CoRR*, abs/2103.13262, 2021. URL https://arxiv.org/abs/2103.13262.

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. doi: 10.1162/neco.1991.3.1.79. URL https://doi.org/10.1162/neco.1991.3.1.79.

Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Comput.*, 6(2):181–214, 1994. doi: 10.1162/neco.1994.6.2.181. URL https://doi.org/10.1162/neco.1994.6.2.181.

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. URL https://arxiv.org/abs/2109.10465.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015a. URL http://arxiv.org/abs/1412.6980.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015b. URL http://arxiv.org/abs/1412.6980.

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. URL https://arxiv.org/abs/2006.16668.

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 (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 6265–6274. PMLR, 2021. URL http://proceedings.mlr.press/v139/lewis21a.html.

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. URL https://arxiv.org/abs/2110.03888.

Yuxuan Lou, Fuzhao Xue, Zangwei Zheng, and Yang You. Cross-token modeling with conditional computation, 2022.

Zixuan Ma, Jiaao He, Jiezhong Qiu, Huanqi Cao, Yuanwei Wang, Zhenbo Sun, Liyan Zheng, Haojie Wang, Shizhi Tang, Tianyu Zheng, Junyang Lin, Guanyu Feng, Zeqiang Huang, Jie Gao, Aohan Zeng, Jianwei Zhang, Runxin Zhong, Tianhui Shi, Sha Liu, Weimin Zheng, Jie Tang, Hongxia Yang, Xin Liu, Jidong Zhai, and Wenguang Chen. Bagualu: targeting brain scale pretrained models with over 37 million cores. In Jaejin Lee, Kunal Agrawal, and Michael F. Spear (eds.), *PPoPP '22: 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Seoul, Republic of Korea, April 2 - 6, 2022*, pp. 192–204. ACM, 2022. doi: 10.1145/3503221.3508417. URL https://doi.org/10.1145/3503221.3508417.

Xiaonan Nie, Pinxue Zhao, Xupeng Miao, and Bin Cui. Hetumoe: An efficient trillion-scale mixture-of-expert distributed training system. *ArXiv*, abs/2203.14685, 2022.

Matt Post. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pp. 186–191, Brussels, Belgium, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-6319. URL https://aclanthology.org/W18-6319.

Ofir Press, Noah A. Smith, and Omer Levy. Improving transformer models by reordering their sublayers. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pp. 2996–3005. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.270. URL https://doi.org/10.18653/v1/2020.acl-main.270.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: memory optimizations toward training trillion parameter models. In Christine Cuicchi, Irene Qualters, and William T. Kramer (eds.), *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, pp. 20. IEEE/ACM, 2020. doi: 10.1109/SC41405.2020.00024. URL https://doi.org/10.1109/SC41405.2020.00024.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation AI scale. *CoRR*, abs/2201.05596, 2022. URL https://arxiv.org/abs/2201.05596.

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. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 8583–8595, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/48237d9f2dea8c74c2a72126cf63d933-Abstract.html.

Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, and Jason Weston. Hash layers for large sparse models. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 17555–17566, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/92bf5e6240737e0326ea59846a83e076-Abstract.html.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016. doi: 10.18653/v1/p16-1162. URL https://doi.org/10.18653/v1/p16-1162.

Noam M. 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. *ArXiv*, abs/1701.06538, 2017.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015. URL http://arxiv.org/abs/1505.00387.

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. URL https://arxiv.org/abs/2105.15082.

Biao Zhang, Philip Williams, Ivan Titov, and Rico Sennrich. Improving massively multilingual neural machine translation and zero-shot translation. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pp. 1628–1639. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.148. URL https://doi.org/10.18653/v1/2020.acl-main.148.

Zhengyan Zhang, Yuxian Gu, Xu Han, Shengqi Chen, Chaojun Xiao, Zhenbo Sun, Yuan Yao, Fanchao Qi, Jian Guan, Pei Ke, Yanzheng Cai, Guoyang Zeng, Zhixing Tan, Zhiyuan Liu, Minlie Huang, Wentao Han, Yang Liu, Xiaoyan Zhu, and Maosong Sun. CPM-2: large-scale cost-effective pre-trained language models. *CoRR*, abs/2106.10715, 2021. URL https://arxiv.org/abs/2106.10715.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. Designing effective sparse expert models. *CoRR*, abs/2202.08906, 2022. URL https://arxiv.org/abs/2202.08906.

Simiao Zuo, Xiaodong Liu, Jian Jiao, Young Jin Kim, Hany Hassan, Ruofei Zhang, Tuo Zhao, and Jianfeng Gao. Taming sparsely activated transformer with stochastic experts. *CoRR*, abs/2110.04260, 2021. URL https://arxiv.org/abs/2110.04260.

APPENDIX

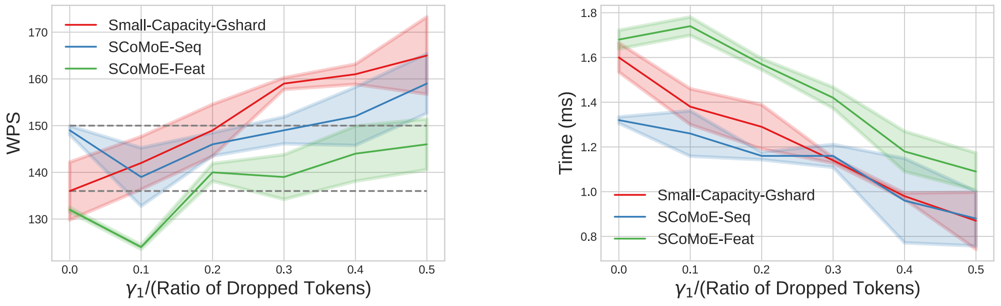
A DETAILS ON EXPERIMENT SETTINGS

Table 3: The statistics of the WMT17-En-Fr and OPUS-100 dataset.

Datasets	Languages	Train	Validation	Test
WMT17-En-Fr	En-Fr	35,762,532	26,854	3003
OPUS-100	100	107,924,846	188×1000	188×1000

Table 4: The configuration of base and large models. Enc/Dec-Layer denotes the number of encoder/decoder layers.

Setting	Enc-Layer	Dec-Layer	d_{model}	Experts	GPU	Nodes	Parameters (B)
Base	6	6	512	8	8	1	0.2
Large	12	12	1024	32	32	4	4.1



(a) (b)

Figure 4: (a) The speed curves of the base models. (b) The communication-time curves of the base models. For Small-Capacity-Gshard, the x-axis is the ratio of dropped tokens while for SCoMoE, it is the ratio of the intra-accelerator communication (γ_1). The bottom dashed line in Figure (a) shows the training speed of Gshard while the top dashed line shows that the three methods have the same training speed at different ratios (SCoMoE-Seq: 0.3, SCoMoE-Feat: 0.5 and Small-Capacity-Gshard: 0.2).

We used the MoE branch of fairseq [10] to implement our MoE models. We prevented the padding tokens from being sent to experts by masking their assignment scores. The capacity factor of the base model was set to 1, while that of the large model was set to 1.25 following the suggestion of (Zoph et al., 2022). The tokens with small assignment scores to an expert were dropped if the expert was out-of-capacity. All models used pre-normalization to stabilize the training. We have found that the MoE models are unstable to train in the large setting, hence we reduced the gradient norm by gradient clipping (i.e., gradient norm was clipped to be no more than 0.5) and gradient accumulation (accumulating the gradients every 4 steps). The gradients of experts were divided by the square root of the number of accelerators, which also reduced the gradient norm and may improve the stabilization of training. We have also found that the MoE model trained in multilingual experiments could not perform well at inference, and that increasing the load-balance loss weight from 0.01 (default value) to 0.1 significantly improves the performance. Therefore, we set the load-balance loss weight to 0.1 in the multilingual experiments, and 0.01 in the bilingual experiments.

The Adam optimizer (Kingma & Ba, 2015a) with learning rate $= 5e^{-4}, \alpha = 0.9, \beta = 0.98$, was used to optimize our models. We employed the inverse-square-root learning schedule (Kingma & Ba, 2015b), with the number of warm-up step being set to 4000. The batch size was set to be 4096 tokens. The dropout rate was 0.3 for bilingual experiments and 0.1 for multilingual experiments. Both the base and large models were trained for 10 epochs. The checkpoint with the best performance on the validation set was selected for testing. In the bilingual experiments, we evaluated the performance on the validation set with BLEU. In the multilingual experiments, the perplexity (ppl) on the validation set was used for faster validation. The BLEU score was computed via sacrebleu (Post, 2018).[11]

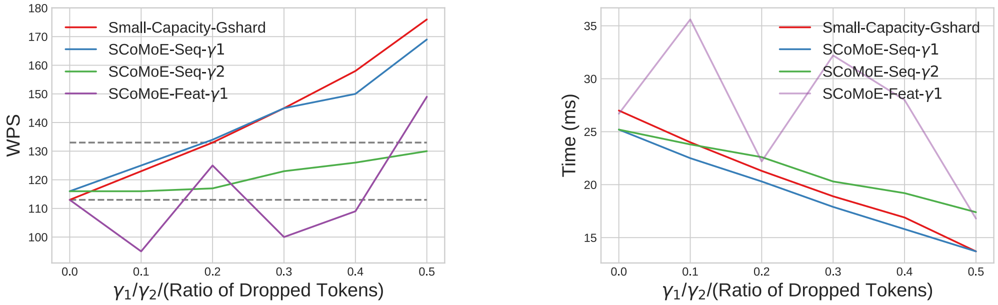
B Evaluation on Training Speed and Communication Time

In this section, we evaluated the training speed and communication time of SCoMoE against Gshard and Small-Capacity-Gshard. We report the training speed in the metric of words-per-second (WPS), which is the number of tokens that the model can process per second, and the time cost of each all-to-all communication in milliseconds.

Base Model We tuned the ratio of the intra-accelerator communication (γ_1) from 0 to 0.5, and report the training speed of SCoMoE at different ratios. In order to compare with Small-Capacity-

[10] `https://github.com/pytorch/fairseq/tree/moe`

[11] The signature of the sacrebleu:BLEU+case.mixed+lang.xx-xx+numrefs.1+smooth.exp+tok.13a+version.1.5.1. xx-xx denotes the language direction, e.g., en-fr



(a)

(b)

Figure 5: (a) The speed curves of the large models. (b) The communication-time curves of the large models. SCoMoE-Seq-γ_1 denotes the SCoMoE-Seq with the intra-accelerator and global communication, for which the x-axis is the ratio of the intra-accelerator communication (γ_1). SCoMoE-Seq-γ_2 denotes the SCoMoE-Seq with the intra-node and global communication, for which the x-axis is the ratio of the intra-node communication (γ_2). The bottom dashed line in Figure (a) shows the training speed of Gshard while the top dashed line shows that the three methods have the same training speed at different ratios (SCoMoE-Seq-γ_1: 0.2, SCoMoE-Seq-γ_2: 0.5, and Small-Capacity-Gshard: 0.2).

Gshard, we also tuned the capacity factor of it. Decreasing the capacity factor is actually dropping tokens during routing. If the capacity factor is set to 0.9, 10% of tokens will be abandoned. We plot the speed and communication-time curves of both SCoMoE and Small-Capacity-Gshard in Figure 4a and 4b. In spite of training with fixed dummy inputs, the training speed of the base models is still unstable. Hence we trained the same model for three times and show the mean and the standard deviation over the three runs in Figure 4a and 4b.

From Figure 4a, we observe that all models can be trained faster as the ratio of (intra-accelerator communication)/(dropped tokens) increases, except that the training of both SCoMoE-Seq and SCoMoE-Feat becomes slower when the ratio is 0.1, but with less communication time than when the ratio is 0. This suggests that the structured communication reduces the communication time but may cause additional overhead. The extra overhead is probably from the computation, since computations in each expert are also split into multiple groups corresponding to multiple communication groups, which is slower than performing all computations as the same time. We also notice that the SCoMoE-Seq is always faster than SCoMoE-Feat, and requires less communication time. This is because that the token clustering is only applied to the SCoMoE-Seq. Even if we set the ratio of the intra-accelerator communication to 0, SCoMoE-Seq is still significantly faster than Gshard. The reason is that the communication cost of token clustering is half as that of the top-2 routing.

It could also be seen that SCoMoE-Seq requires less communication time than Small-Capacity-Gshard with the same ratio, but the training speed of SCoMoE-Seq is always slower. This is because that SCoMoE forces some tokens to be processed at local devices, while Small-Capacity-Gshard simply drops the same number of tokens, which saves computation time.

Large Model We could tune both the ratio of the intra-accelerator (γ_1) and intra-node communication (γ_2) for large SCoMoE models. But it is expensive to tune them at the same time. Therefore, we fix one ratio, and tune the other. The curves of training speed and communication time are visualized in Figure 5a and 5b respectively. Clearly, SCoMoE-Seq with the intra-accelerator communication (SCoMoE-Seq-γ_1) can be trained in a speed that is comparable to that of Small-Capacity-GShard, consuming the amount of communication time which is also comparable to that of Small-Capacity-GShard. Its training speed is much faster than that of SCoMoE-Seq with the intra-node communication. This is not surprising as intra-accelerator communication is much faster than the intra-node communication. However, both the training speed and the amount of communication time spent by SCoMoE-Feat-γ_1 are unstable, which are faster/less than those of Gshard when the ratio is 0.2 and 0.5, but slower/more for other ratios. The reason for this might be that the amount of communication time is related to the shape of communicated tensor.

Table 5: The translation performance and training speed of Gshard and SCoMoE-Seq with/without token clustering and layer reordering. ✓/ × denotes with/without. Clustering denotes the token clustering. Reordering denotes changing the order of layers according to the architecture in Figure 2

Model	Clustering	Reordering	Multilingual	Bilingual	Speed
Gshard	×	×	24.41	37.96	1x
Gshard	×	✓	22.73	38.29	1x
Gshard	✓	✓	22.86	37.93	1.1x
SCoMoE-Seq	×	×	22.81	38.0	1.05x
SCoMoE-Seq	✓	✓	**24.45**	**38.24**	1.1x

Theoretic Analysis on Communication Time We have empirically evaluated the training speed and the amount of communication time of SCoMoE by varying γ_1 and γ_2. The bandwidth of communication has a great impact on these empirical results in practice. To complement the empirical evaluation, we further theoretically analyze the amount of time required by the all-to-all communication and estimate speedups that can be achieved by the proposed structured communication method. We estimate the all-to-all communication time as the combination of the intra-node, and inter-node communication time as the time cost for the intra-accelerator communication (memory copy) is negligible in comparison to the other two communication channels. Suppose that the total amount of data for communication is D MB, and the number of accelerators in one node is m. Given that, the amount of data for the intra-/inter-node communication is $\frac{D}{m}$/$\frac{D(m-1)}{m}$. Suppose that the bandwidth of the intra/inter-node communication is B_1/B_2 MB/s. The amount of the all-to-all communication time (seconds) can be estimated as follows:

$$\begin{aligned} T_{\text{all-to-all}} &= \frac{D}{mB_1} + \frac{D(m-1)}{mB_2} \\ &= \frac{D(m-1)}{m}(\frac{1}{(m-1)B_1} + \frac{1}{B_2}) \end{aligned} \tag{7}$$

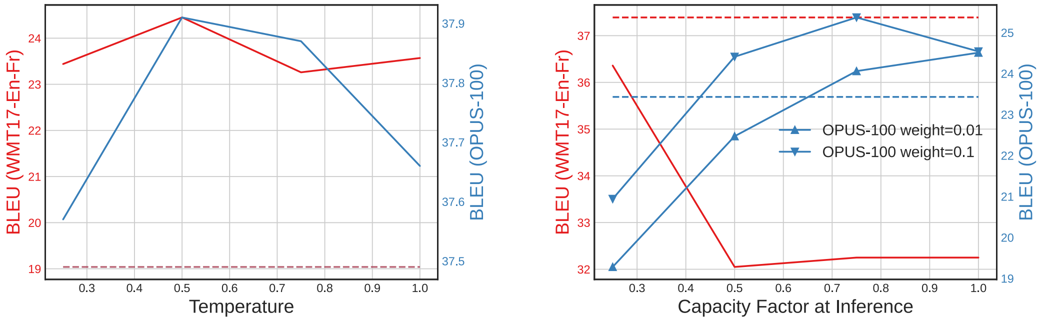
According to the equation, increasing the bandwidth of the inter-node communication (B_2) is more beneficial than increasing the bandwidth of the intra-node communication (B_1) as $\frac{1}{m-1} \leq 1$. If we set the ratio of the intra-node communication to γ_2 (γ_1=0, 0¡γ_2¡1), the amount of the all-to-all communication time can be computed as:

$$\begin{aligned} T_{\gamma_2} &= (1-\gamma_2)T_{\text{all-to-all}} + \gamma_2\frac{D}{B_1} \\ &= (1-\gamma_2)(\frac{D}{mB_1} + \frac{D(m-1)}{mB_2}) + \gamma_2\frac{D}{B_1} \\ &= (1-\gamma_2)\frac{D(m-1)}{m}(\frac{1}{B_2} - \frac{1}{B_1}) + \frac{D}{B_1} \end{aligned} \tag{8}$$

The absolute value of the derivative of γ_2 is proportional to $\frac{1}{B_2} - \frac{1}{B_1}$. Either decreasing B_2 or increasing B_1 would make the SCoMoE more efficient. Therefore, If the bandwidth of the intra-node communication is slow, it is better to set $\gamma_2 = 0$. Typically, there are two types of intra-node connections with different bandwidths: PCIE and NVlink/NVSwitch. The NVlink/NVSwitch connection is much faster than the PCIE connection. Furthermore, the network interface controller (NIC) is also connected with accelerators through PCIE. More communications sharing PCIE connections, will make the communication slower. Therefore, we recommend to set $\gamma_2 > 0, \gamma_1 = 0$ if fast intra-node connections with NVlink/NVSwitch are available, and $\gamma_1 > 0, \gamma 2 = 0$ if slow intra-node connections with PCIE are used.

C EFFECTS OF TOKEN CLUSTERING

We compared the translation performance and training speed of Gshard and SCoMoE with vs. without token clustering. Results are shown in Table 5. It could be seen that the Gshard with token



(a)

(b)

Figure 6: (a) The translation performance of SCoMoE-Seq with different temperatures on the test set of OPUS-100 and WMT17-En-Fr. The horizontal dashed line denotes the BLEU scores of the SCoMoE-Seq with the sigmoid gate on OPUS-100 and WMT17-En-Fr. (b) The translation performance of Gshard with different eval-capacity-factor. The two horizontal dashed lines denote the BLEU scores of the standard Transformer on OPUS-100 (blue) and WMT17-En-Fr (red). The two dashed lines in Figure (a) are overlapped together.

clustering or layer reordering performs comparably to Gshard without using them on bilingual translation, but is significantly worse than the original GShard on multilingual translation. Results of SCoMoE demonstrate that token clustering can significantly benefit our model on both bilingual and multilingual machine translation in terms of both BLEU and training speed.

D Importance of Differentiable Sorting

We conducted experiments to investigate how the temperature of softmax affect the performance. We compared the translation performance of SCoMoE-Seq with different temperatures. Results are plotted in Figure 6a, from which we could see that the performance of the model is sensitive to the temperature. We also report the performance of SCoMoE-Seq with the sigmoid gate as the baseline, which is also displayed in Figure 6a (the horizontal dashed line). It could be seen that SCoMoE-Seq with the softmax gate is consistently better than that with the sigmoid gate at different temperatures.

E Effect of Capacity Factor at Inference

The capacity factor is of importance to the training of MoE models (Yang et al., 2021). In our multilingual NMT experiments, we have found that the models trained on the OPUS-100 dataset performs poorly on the test set, unless with a high capacity factor, substantially higher than that used during training. For convenience, we refer to the capacity factor at inference as eval-capacity-factor. The capacity during training and inference can be computed as follows:

$$\text{capacity}_{\text{train}} = \frac{2 \times \text{number of tokens}}{\text{number of experts}} \times \text{capacity-factor} \tag{9}$$

$$\text{capacity}_{\text{eval}} = \text{number of tokens} \times \text{eval-capacity-factor} \tag{10}$$

We evaluated the trained Gshard model with different eval-capacity-factors and show the performance in Figure 6b. The model for evaluation was trained with the capacity factor of 1.0. The performance of Gshard is even worse than the standard Transformer, when the eval-capacity-factor is smaller than 0.75. Setting eval-capacity-factor=0.75 is equivalent to setting capacity-facor=3.0 if the number of experts is 8, which is very expensive. The costs for communication, computation and memory are all proportional to the capacity factor, hence running Gshard on the OPUS-100 dataset is expensive. We also show the impact of eval-capacity-factor on the translation performance on the WMT17-En-Fr dataset in Figure 6b. It is interesting that the performance of Gshard on WMT17-En-Fr decreases as the eval-capacity-factor increases, which is opposite to the observation on OPUS-100. Increasing the eval-capacity-factor changes the routing of MoE, which may hurt the performance. But why do the the multilingual MoE models require a large capacity? We conjecture that the routing of multilingual MoE is language-specific, i.e., the tokens in related languages are routed to the same experts. In our experiments, each batch contains multiple languages during training, but only one language at inference. Therefore, sentences in the same language are routed to the same experts at inference, which causes these experts to overflow and tokens to be discarded. To validate our hypothesis and alleviate this problem, we increase the load-balance weight in Gshard from 0.01 to 0.1, and observe significant improvements in Figure 6b. But we still need a high capacity factor for the inference of multilingual MoE models. We leave the issue to our future work.