ByteDance | Seed

Comet: Fine-grained Computation-communication Overlapping for Mixture-of-Experts

Shulai Zhang[1,2,○,*], **Ningxin Zheng**[1,○,†], **Haibin Lin**[1,†], **Ziheng Jiang**[1], **Wenlei Bao**[1], **Chengquan Jiang**[1], **Qi Hou**[1], **Weihao Cui**[2], **Size Zheng**[1], **Li-Wen Chang**[1], **Quan Chen**[2,†], **Xin Liu**[1,†]

[1]ByteDance Seed, [2]Shanghai Jiao Tong University

[○]Equal Contribution, [*]Work done at ByteDance Seed, [†]Corresponding authors

Abstract

Mixture-of-experts (MoE) has been extensively employed to scale large language models to trillion-plus parameters while maintaining a fixed computational cost. The development of large MoE models in the distributed scenario encounters the problem of large communication overhead. The inter-device communication of a MoE layer can occupy 47% time of the entire model execution with popular models and frameworks. Therefore, existing methods suggest the communication in a MoE layer to be pipelined with the computation for overlapping. However, these coarse grained overlapping schemes introduce a notable impairment of computational efficiency and the latency concealing is sub-optimal.

To this end, we present Comet, an optimized MoE system with fine-grained communication-computation overlapping. Leveraging data dependency analysis and task rescheduling, Comet achieves precise fine-grained overlapping of communication and computation. Through adaptive workload assignment, Comet effectively eliminates fine-grained communication bottlenecks and enhances its adaptability across various scenarios. Our evaluation shows that Comet accelerates the execution of a single MoE layer by 1.96× and for end-to-end execution, Comet delivers a 1.71× speedup on average. Comet has been adopted in the production environment of clusters with ten-thousand-scale of GPUs, achieving savings of millions of GPU hours.

Correspondence: Ningxin Zheng, Haibin Lin, Quan Chen, Xin Liu

Project Page: https://github.com/bytedance/flux

1 Introduction

Recent advancements in large language models have revolutionized multiple domains, including natural language processing [35, 36], computer vision [16] and multi-modal perception [3, 14]. These achievements demonstrate that scaling up model size can significantly enhance model capacity. However, the growth in model parameters poses substantial challenges for the deployment of such giant models, as computational resources increasingly constrain model capacity [28].

To this end, Mixture-of-Experts (MoE) [29] introduces a sparse structure, within which only part of the parameters is activated. Instead of interacting with all parameters in dense models, MoE models allow each input to interact with only a few experts. For example, the Mixtral-8x7B model [12] comprises 45 billion parameters in total, while only 14 billion parameters are active during runtime. Nowadays, MoE has emerged as a key architecture for scaling

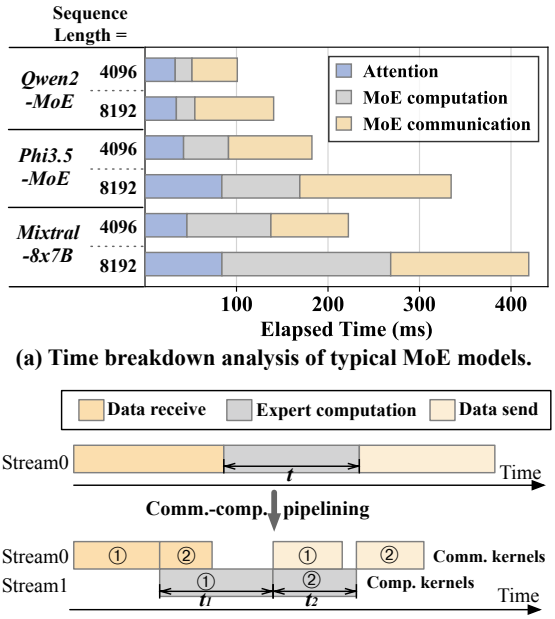

Figure 1 Analysis of the execution of MoE. (a) Time breakdown of MoE models executed on 8 H800 GPUs using Megatron-LM. (b) An illustration of communication-computation overlap by partitioning an expert computation kernel into two.

models to trillion-plus parameters.

The increase in parameter size in MoE models allows for the integration of greater amounts of information, but it poses challenges in expert placement. A typical approach is to distribute the experts across different GPUs as a single GPU cannot store all experts [13]. Consequently, during the execution of MoE layers, there is an intensive need for data exchange among GPUs. In the forward pass of several popular MoE models, the communication among devices accounts for 47% of the total execution time on average, as shown in Figure 1(a).

In a distributed environment, executing an MoE layer involves data reception, expert computation, and data transmission, as depicted in in Figure 1(b). To reduce communication overhead, one effective strategy is to pipeline the process, overlapping communication with expert computation [8, 10, 31, 32]. This approach involves partitioning input data into smaller data chunks, allowing decomposed communication and computation phases to overlap. In the example in Figure 1(b), the received input data is divided into two chunks, and this coarse-grained overlapping reduces the overall execution time relative to non-pipelined execution.

The overlapping in existing mechanisms remains suboptimal due to two primary inefficiencies. First, the efficiency of partitioned experts declines as the data chunks assigned to each expert become smaller, potentially leading to under-utilization of GPU computational resources (e.g., the total compute time of experts after partitioning $t_1 + t_2$ exceeds the original time t). The coarse-grained partitioning results in unavoidable GPU idle time during the initial and final communication phases, such as when receiving data for chunk 1 and sending data for chunk 2, which do not overlap with computation. Consequently, minimizing the non-overlapping time in these phases while maintaining computational efficiency is crucial. This is challenging because the data dependency between communication and computation is complex and it is hard to be overlapped in a fine-grained granularity efficiently. Second, due to the dynamic nature of MoE, the input shapes for experts are various at runtime, thereby posing diverse communication and computation burdens on GPUs. Encapsulating communication and computation tasks into separate kernels on different streams, like almost all the prior researches do, restricts control over hardware resources and results in non-deterministic kernel performance, thereby hindering seamless overlap (e.g., the computation of chunk 1 and the receiving of chunk 2 are misaligned). The second challenge, therefore, is to dynamically ensure precise allocation of hardware resources between computation and communication workloads at runtime.

The complex data dependency, and the dynamic computation and communication workloads in MoE impede existing systems to realize efficient communication-computation overlap. We therefore propose COMET, a system that enables fine-grained communication-computation overlapping for efficient MoE execution. COMET introduces two key designs: 1) A dependency resolving method that identifies complex data dependencies between communication and computation operations in MoE, enabling optimized computation-communication pipeline structuring. 2) An adaptive workload assignment method that dynamically allocates GPU thread blocks to different workloads within a kernel, balancing communication and computation to improve latency concealment.

COMET facilitates fine-grained overlapping in MoE by analyzing shared data buffers between communication and computation operations, referred to as *shared tensor*. By decomposing the shared tensors along specific dimensions and reorganizing tensor data along with intra-operator execution order,

COMET eliminates the granularity mismatches between communication and computation, thereby enabling fine-grained overlapping. To ensure precise resource allocation and effective latency concealment, COMET integrates communication and computation tasks within fused GPU kernels. Through thread block specialization, COMET isolates the impact of communication on computation performance , maintaining high computational efficiency. By adjusting the number of thread blocks allocated to each workload, COMET effectively balances communication and computation latencies and reduces bubbles in overlapping.

We have integrated COMET into Megatron-LM [33] and verified the capability of COMET with various parallel strategies. Our extensive experiments on Nvidia H800 and L20 clusters show that COMET delivers 1.96× speedup for typical MoE layers, and 1.71× speedup for end-to-end MoE model execution (Mixtral-8x7B [12], Qwen2-MoE [2], Phi3.5-MoE [1]) on average, compared with the SOTA MoE systems. COMET has been deployed to accelerate training and inference of large MoE models in production clusters comprising over ten thousand GPUs, achieving savings of millions of GPU hours. COMET introduces a fine-grained pipelined programming model for computation and communication. **We will open-source COMET, aiming to inspire further optimizations**, such as implementing the programming model in COMET using compilers like Triton [26] or TVM [6].

2 Background and Motivation

2.1 MoE Structure

Table 1 Description of symbols.

Symbol	Description
L	Number of transformer layers
E	Total number of experts
$topk$	Number of experts that each token is routed to
TP	Tensor parallel size
EP	Expert parallel size
W	Total parallel world size ($TP\times$ EP)
M	Input token length × Batch size
N	Embedding size of a token
K	Hidden size of the feed-forward layer in experts

Mixture of Experts (MoE) is critical for efficiently scaling models. By enabling sparse activation of parameters, MoE allows for the integration of more parameters without increasing execution costs, thereby enhancing performance. The key idea of MoE is that it consists of multiple small models, namely *experts*

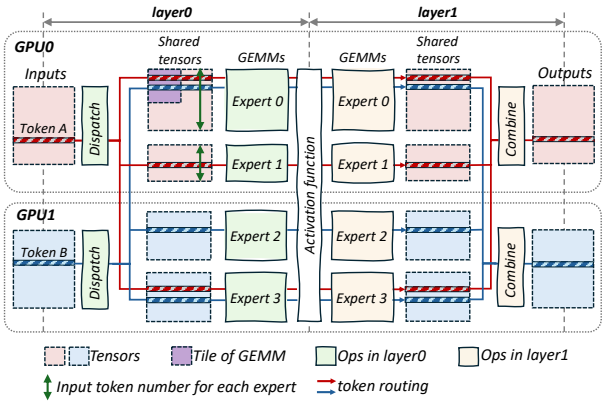

Figure 2 Example of an MoE layer across two GPUs, with two experts reside on GPU0 and two reside on GPU1. The MoE layer is composed of two feed-forward layers. In this example, for each token in the input buffer, it is dispatched to three experts ($topk = 3$) in layer0 and then the results are combined in layer1. The shape of experts is $N \times K$ in layer0 and $K \times N$ in layer1.

and tokens are only routed to partial experts for computation. Figure 2 shows the typical execution flow of an MoE layer and Table 1 explains symbols to describe the execution of an MoE model.

Each input token is assigned to one or more experts for computation, with assignments determined by various algorithms [15, 40, 41]. A common method involves a gate network [29] that selects the *topk* experts for each token, as shown in Figure 2, where token A is routed to Expert0, Expert1 and Expert3. After passing through two feed-forward layers of General Matrix Multiply (GEMM), the *topk* outputs are gathered and reduced to produce the final result.

The operations in MoE's layer0 comprise token communication (dispatch) across GPUs and the first layer of expert computations (GEMM operations), thereby establishing a communication-computation pipeline. MoE's layer1 includes the second layer of expert computations, token undispatch and the topk reduction (combine), forming a computation-communication pipeline.

MoE employs two primary parallelization strategies: **Expert parallelism** [13] and **Tensor parallelism** [33]. In expert parallelism, the weights of different experts are distributed across separate GPUs, with each expert's weights being fully intact. Tokens are routed to the corresponding devices of their respective experts. Figure 2 shows a case for expert parallelism, with Expert0 and Expert1 reside on GPU0 and others reside on GPU1. In contrast, tensor parallelism

partitions the weights of all experts along the hidden dimension, with each GPU hosting a portion of the weights from all experts. Both expert and tensor parallelism are essential for the efficient execution of MoE. In practical deployment of MoE models, a hybrid parallelism approach combining both expert and tensor parallelism is often applied.

2.2 Computation and Communication Overlapping

As the MoE architecture grows larger and sparser, the proportion of time spent on communication in MoE models becomes increasingly significant, as shown in Figure 1(a). As illustrated in section 1, coarse-grained overlapping of computation and communication offers limited optimization potential, and kernel-level scheduling is not efficient for dynamic workloads. Thus, it is more efficient to perform the overlapping at a fine-grained granularity (such as token-wise) and integrates computation and communication workloads into fused GPU kernels. Adopting such a finer-grained overlapping could extremely unleash further optimization opportunities. However, achieving such fine-grained overlapping in MoE is non-trivial and there are two primary obstacles in our observation.

2.2.1 Granularity mismatch between computation and communication

In MoE systems, the token serves as the fundamental unit of data movement, illustrated by the movement of Token A in Figure 2. To maximize GPU compute efficiency, high-performance GEMM(GroupGEMM) kernels typically organize rows into tiles for processing. The purple block in Figure 2 represents such a computation tile in GEMM kernels, exemplified by a 128x128 tile. Therefore, the GEMM computations associated with a single expert may require 128 tokens distributed across multiple GPUs. When fusing computation and communication at fine granularity, the disparity between token-level data transfer and tile-level computation introduces considerable challenges: The complex data dependency adversely affects the efficiency of overlap, prompting the use of fine-grained communication, while integrating fine-grained communication with computation within fused kernels is also challenging.

Complex data dependency. The tokens needed for each computation tile, determined by the MoE's gate at runtime, are randomly distributed across multiple devices. Computation for a tile cannot start until all required tokens are available. As shown in Figure 2, Expert0's tile does not initiate processing until both Token A and Token B are received. Thus, with coarse-grained data communication, data preparation time for each computational tile may be prolonged because of this irregular and complicated data dependency. To mitigate this, we should employ fine-grained communication, where each computational tile reads or writes only the data it requires directly through the Unified Virtual Address [18], and leverage the data reorganization and rescheduling to hide it with computation efficiently.

Fine-grained communication. The integration of token-wise communication with tile-wise computation for overlapping is non-trivial. Remote I/O operations between GPUs exhibit significantly higher latency compared to local GPU memory access. Therefore, executing numerous fine-grained read and write operations on remote data tokens within computation thread blocks can block subsequent computational tasks, leading to a significant decline in kernel efficiency. This challenge is especially evident in the Hopper architecture, where computation kernels leverage Tensor Memory Accelerator (TMA) hardware instructions [20] to establish asynchronous compute pipelines. The integration of long-latency remote I/O operations within these asynchronous pipelines can considerably prolong the overall execution time, adversely affecting performance. Thus, it is critical to constrain the impact of fine-grained communication on computation kernels.

Our first insight is that resolving the granularity mismatch between computation and communication in MoE models is the key to enable efficient overlap of these two processes.

2.2.2 Diverse loads of computation and communication

Another characteristic of MoE is the dynamic routing of tokens to different experts, resulting in varying input shapes for experts at runtime (e.g., the token number received by Expert0 and Expert1 are different as shown in Figure 2). This variability imposes differing communication and computation demands on GPUs. Besides, the hardware environments can also have various compute architectures or network topologies, providing different compute capacities and communication bandwidths. Achieving seamless overlap between computation and communication thus requires dynamically adjusting the allocation of GPU resources to different workloads, which is hard to be realized through wrapping workloads into separate kernels.

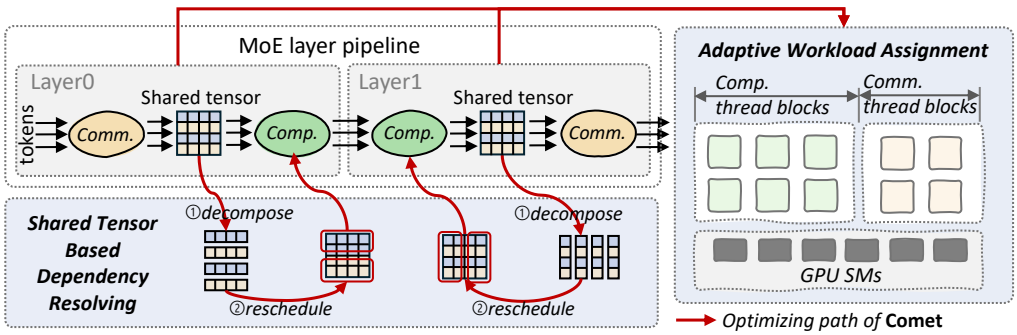

Figure 3 Design overview of COMET. COMET is composed of a shared tensor-based dependency resolving method and an adaptive workload assignment mechanism.

Our second insight is that the resource allocation should be adaptive within kernels at runtime to further achieve seamless communication-computation overlapping.

3 Design of Comet

In this section, we present the core design of COMET, a Mixture of Experts (MoE) system optimized for efficient execution of MoE layers through pipelined execution and fine-grained overlapping of communication and computation. Our analysis reveals that the MoE architecture has two distinct producer-consumer pipelines: the communication-computation pipeline and the computation-communication pipeline, as illustrated in Figure 3. Tokens traverse the pipelines as depicted and the operations within each pipeline are linked through a shared buffer, referred to as the **shared tensor**, serving as both the producer's output buffer and the consumer's input buffer. To minimize overall latency and enhance pipeline performance, COMET introduces two key mechanisms aimed at overlapping computation and communication workloads effectively.

1. Shared tensor based dependency resolving: As previously mentioned, the intricate data dependencies between communication and computation pose a challenge to achieving seamless overlap between these operations. To address this, we examine the data dependencies by analyzing the shared tensor. Our analysis reveals that the shared tensor can be decomposed, and the associated computations can be rescheduled to overlap more effectively with communication. Accordingly, the dependency resolving process employs two key optimization strategies on the shared tensors as shown in Figure 3: ① Decomposing the shared tensors along specific dimensions to break the coarse-grained data dependencies and, ② rescheduling the computations to enhance efficiency while ensuring effective overlapping.

2. Adaptive workload assignment: Following pipeline optimization by the dependency resolving, the pattern of communication-computation overlap becomes more consistent and regular. To effectively hide the fine-grained communication latency, it is essential to allocate appropriate hardware resources to both communication and computation workloads. Given that these workloads exhibit different performance characteristics depending on input shapes, model configurations, and hardware environments, the adaptive workload assignment scheme dynamically balances computation and communication. This approach generates highly efficient horizontally-fused kernels for the MoE system, thereby optimizing latency concealment.

As shown in Figure 3, COMET first leverages the shared tensor based dependency resolving method to optimize the pipelines in the MoE structure by decomposing and rescheduling the shared tensors. According to the reformed pipelines, COMET then provides highly-efficient fused kernels through the adaptive workload assignment mechanism.

3.1 Shared Tensor Based Dependency Resolving

We now introduce how to resolve the complex data dependency between computation and communication in MoE. It aims to bridge the granularity of communication and computation operations to sustain high efficiency by decomposing and rescheduling shared tensors.

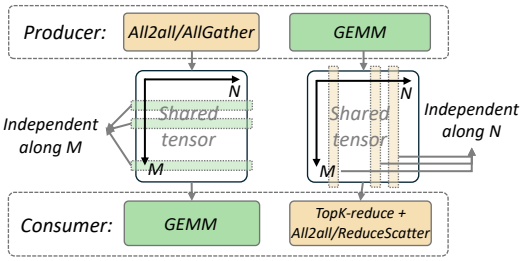

Figure 4 The producer-consumer modeling of layer0 (left) and layer1 (right) of an MoE layer. The global size of the shared tensor is $(M \times topk, N)$ for both layer0 and layer1.

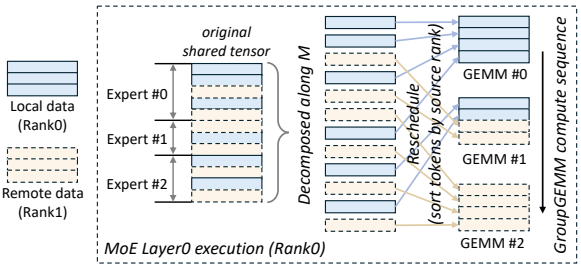

Figure 5 Decompose and reschedule the shared tensor in MoE layer0. In this illustration, three experts are located on Rank 0, each requiring both local and remote data for computation.

3.1.1 How to decompose the shared tensor?

Shared tensors, as the bridge between the producer operator and the consumer operator, is the key to enable overlapping. Notably, overlapping can occur only when the producer and consumer operate on independent data within the shared tensor, as illustrated in Figure 4. Thus, we analyze the access pattern of operators on the shared tensor and decompose it along a specific dimension where data remain independent for the consumer operator.

For example, in the communication-computation pipeline in layer0, the consumer operator is a GEMM, with the shared tensor serving as its input matrix. In this case, tokens are independent with each other alongside the M (token) dimension, allowing for decomposition of the shared tensor along M. However, since the computation of a GEMM tile involves multiplication and reduction along the token embedding dimension to produce the final outputs, decomposing the shared tensor along this dimension is not feasible.

As for the computation-communication pipeline in layer1, the consumer operator contains a top-K reduction, which reduces tokens along the M dimension, leading to significant interdependencies between tokens along this dimension. Thus, the shared tensor can only be decomposed along the N dimension where elements are independent.

3.1.2 How to reschedule the decomposed shared tensor?

At the finest granularity, the shared tensor can be split into individual rows or columns, enabling the consumer to begin computation as soon as a single row or column is received. However, this level of granularity results in low computational efficiency, particularly in pipelines involving compute-intensive GEMMs, which are typically organized and processed in tiles to achieve high utilization. Therefore, after decomposing shared tensors along specific dimensions, the resulting sub-tensors must be reorganized and rescheduled into tiles for computation. The rescheduling of shared tensors follows two principles: ① Rescheduled sub-tensors should align with the original computation tile granularity for computational efficiency. ② The scheduling policy should prioritize portions of the producer that can be immediately used by the consumer, allowing the consumer to begin execution as early as possible.

Comet leverages GroupGEMM to perform the computations for all experts on current rank. In the communication-computation pipeline (MoE layer0), the shared tensor, consumed by GroupGEMM, is decomposed along the M dimension. To enable early computation by the experts, tokens are sorted based on their source rank, as shown in Figure 5. The compute sequence of tiles in the GroupGEMM is then designed to minimize dependency on remote data, with computation beginning from tiles containing local tokens while the transfer of other remote tokens proceeds concurrently.

In the computation-communication pipeline (MoE layer1), the shared tensor undergoes a top-k reduction after processing by the GroupGEMM of experts. As analyzed previously, the shared tensor is decomposed along the N dimension. The tile computation sequence is adjusted (Figure 6) to enable the consumer operator to start processing before expert computations are fully completed. Instead of computing each expert sequentially, GroupGEMM operations are executed column-wise. This approach allows the reduction and communicate operations to proceed as soon as the first T_N columns of the shared tensors are computed. Without rescheduling, tokens could only be reduced after all experts have completed their computations.

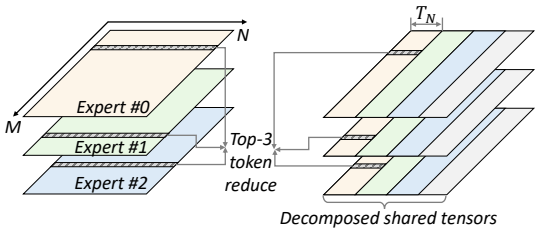

Figure 6 Rescheduled compute sequence for MoE layer1 ($E = 3$ and $topk = 3$). The execution order of the GroupGEMM is indicated by color (yellow → green → blue → grey). Here, T_N denotes the tile size of a GroupGEMM along the N dimension.

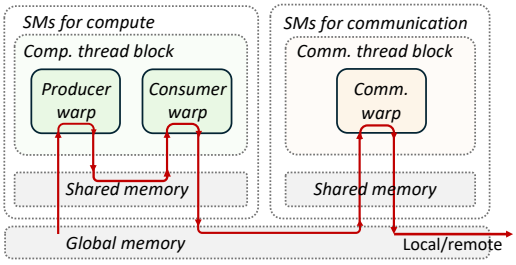

Figure 7 Kernel design for the MoE layer1 on Hopper architecture. Each SM only accommodate one thread block. The red arrows indicates the route of data movement.

3.2 Adaptive Workload Assignment

With the decomposition and rescheduling of shared tensors, the pipelines in MoE can now achieve fine-grained overlap. To ensure effective latency hiding, the durations of fine-grained communication and computation must be closely aligned to minimize pipeline bubbles. Achieving this requires adaptive resource allocation for both computation and communication, tailored to specific tasks involved.

3.2.1 Thread block specialization

A straightforward approach to achieve communication-computation overlap in Mixture of Experts (MoE) is to encapsulate the entire pipeline within homogeneous thread blocks, integrating communication I/O into the prologue or epilogue of the computation (GEMM), a strategy referred to here as vertical fusion. Through vertical fusion, thread blocks execute concurrently, but the overlap occurs irregularly, leading to non-deterministic latencies of communication and computation, making it challenging to balance their durations for latency hiding. Furthermore, token-level fine-grained I/O in MoE can significantly reduce the computational efficiency of the underlying kernels, particularly on advanced architectures such as Hopper. To address this, we implement thread block-level isolation between communication and computation workloads. This isolation enables precise control over hardware resource allocation for each workload, facilitating a balanced distribution between computation and communication that maximizes latency hiding.

Figure 7 depicts the details of the thread block specialized kernel on Hopper, with the critical data path highlighted in red. Due to the isolation between communication and computation, the GEMM thread blocks in COMET utilize the same implementation as the default GEMM before fusion. In the scenario depicted in Figure 7, where the GEMM is compiled using CUTLASS on the Hopper architecture, the GEMM execution is distributed across different warps. Specifically, the producer warp loads data from global memory into a shared memory buffer with the async TMA instructions, while the consumer warp initiates tensor core MMA operations [21]. The communication thread blocks subsequently read the results produced by the consumer warp from global memory. Following the top-K reduction, the warps within the communication blocks either write tokens to the local global memory or transmit them to remote destinations. This thread block-specialized programming model is easily portable to other architectures, such as Ampere and Volta, requiring only a substitution of the respective compute thread block implementation.

Hardware resource restriction. The proposed thread block-specialized kernel is designed with the primary objective of minimizing data movement costs. However, this design must also contend with hardware resource limitations. For instance, it is theoretically feasible to integrate communication warps with computation warps within the same thread block to eliminate redundant global memory accesses. However, the thread number restriction of warps constrict the communication operator to fully utilize the communication bandwidth. From another perspective, the warps for communication also interfere with the computation warps within the same thread block.

3.2.2 Adaptive thread block assignment

Suppose that there are n thread blocks for the fused kernel, within which n_p blocks serve as producers in the pipeline and n_c blocks serve as consumers. Identifying an optimal division point n_p/n_c is crucial for maximizing overall efficiency. We demonstrate that the optimal division point is influenced by the shape of input and specific model configurations in an MoE layer. To investigate this, we measure the duration

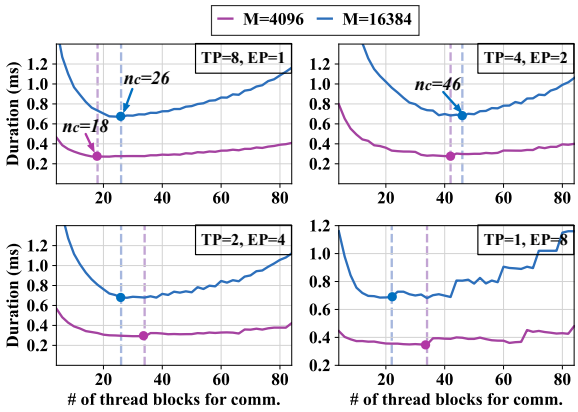

Figure 8 Duration of the MoE layer1 kernel with varying number of thread blocks assigned for communication (n_c). The total number of thread blocks is identical to the number of SMs on Hopper(132). The figure shows four cases with different parallelisms.

of MoE layer1 across various input sequence lengths and parallelization strategies, as shown in Figure 8. It is observed that there exist an optimal division point under different configurations.

When the input token length changes, although the data sizes processed by communication and computation operations both scale with input length, the scalability of the respective resource requirements differs. Consequently, the optimal division point shifts with changes in input length. For example, when $TP = 8$, the optimal n_c changes from 18 to 26 when M is changed from 4096 to 16384. When the model configuration (parallel strategy) is modified, the optimal division point undergoes a significant alteration. For instance, when TP is adjusted from 8 to 4, the optimal n_c is transformed from 26 to 46 with $M = 16384$.

COMET's library comprises multiple pre-compiled kernels, each with a distinct division point. Prior to deployment, the optimal configuration for each setup is profiled and stored as metadata. During runtime, COMET utilizes this metadata to select the optimal kernel for execution.

4 Implementation

COMET consists of approximately 12k lines of C++ and CUDA code and 2k lines of Python. COMET provides a suite of user-friendly Python APIs and developers can seamlessly integrate the APIs into their frameworks. In production environment, COMET has been implemented in Megatron-LM for large-scale MoE training. The source code will be available on GitHub.

Table 2 Configuration of MoE models used in experiments. The models are open-sourced on Hugging Face [9]. The meaning of symbols are explained in Table 1.

	L	E	topk	N	K
Mixtral 8x7B	32	8	2	4096	14336
Qwen2-MoE-2.7B	24	64	4	2048	1408
Phi-3.5-MoE	32	16	2	4096	6400

Optimized GEMM kernels for MoE. COMET extensively utilizes the programming templates provided by CUTLASS to generate highly efficient GEMM kernels. Additionally, it incorporates various optimizations to minimize data movement overhead. For instance, in MoE layer 0, the row indices of the input matrix for GEMM operations must be accessed from global memory at each K iteration. By caching these row indices in registers, COMET significantly reduces the global memory access cost.

NVSHMEM as communication library. We employ NVSHMEM [24] within kernels to support fine-grained communication. NVSHMEM is a communication library designed for NVIDIA GPUs. It creates a global address space for data that spans the memory of multiple GPUs and can be accessed with fine-grained GPU-initiated operations and CPU-initiated operations. Unlike NCCL [23], which targets high-level communication operations, NVSHMEM offers a more composable, low-level API that facilitates finer data access granularity within kernels.

5 Evaluation

5.1 Experimental Setup

Testbed. We evaluate COMET on a server equipped with 8 Nvidia H800 GPUs (80 GB memory each). These GPUs are interconnected through NVLink. Our software environment includes CUDA 12.3, NVSHMEM 2.11, Pytorch 2.4.0 and Megatron-LM (git-hash 6dbe4c).

Comparing targets. We then compare COMET with several baselines. All baselines are implemented on Megatron-LM, which is a widely adopted framework for high-performance model execution, integrating hybrid parallel strategies.

The baselines are: (a) **Megatron-Cutlass**: Megatron with MoE experts that are implemented through CUTLASS grouped GEMM [22]. (b) **Megatron-TE**: Megatron with experts that use transformer en-

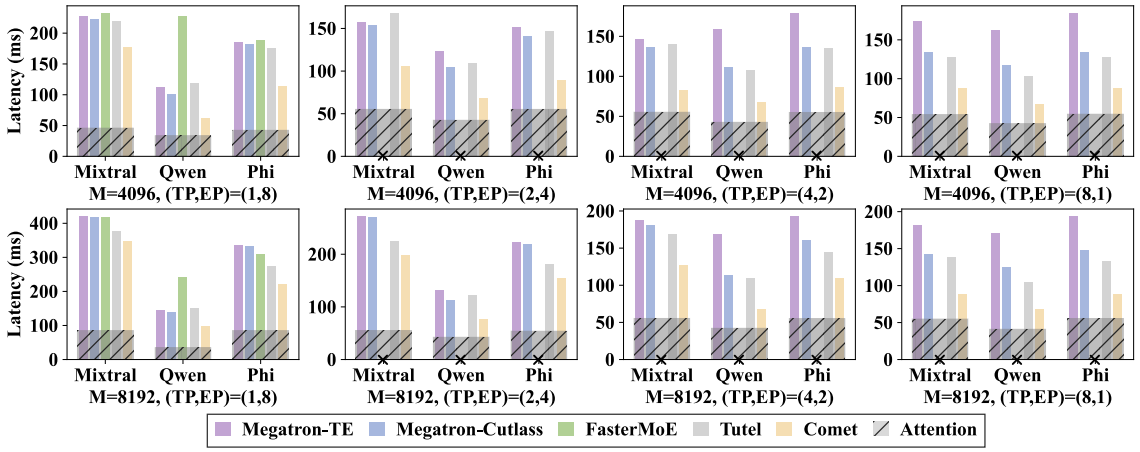

Figure 9 End-to-end MoE model latency. For the computation of MoE layers, the number of token on each device before permutation is $M \times W/TP$. The hatched region represents the identical duration of non-MoE (attention) layers in different mechanisms. Note that FasterMoE only supports expert parallelism for MoE layers.

gine [25]. Transformer Engine is Nvidia's library for accelerating transformer models on NVIDIA GPUs. (c) **FasterMoE** [7, 8]: FasterMoE is an MoE system that customizes All-to-All communication to overlap the communication and computation operations of experts. (d) **Tutel** [10]: Tutel delivers several optimization techniques for efficient and adaptive MoE, including adaptive parallelism, the 2-dimensional hierarchical All-to-All algorithm and fast encode/decode with sparse computation on GPU.

5.2 Overall Performance

We evaluate the end-to-end performance of Comet in multiple large MoE models, including Mixtral 8x7B [12], Qwen2-MoE [2] and Phi3.5-MoE [1]. The configurations of these models are shown in Table 2. The experiment is conducted with various input token lengths and diverse hybrid parallel strategies. The experimental details and results are shown in Figure 9. Note that when $TP < W$, Megatron-LM enables data parallelism for non-MoE layers to improve overall throughput and the data parallel size is W/TP. The computation of attention layers are identical with different mechanisms using Megatron-LM, and only the MoE layer is implemented differently with diverse mechanisms.

As observed, the end-to-end latencies of the benchmarks are reduced by 34.1%, 42.6%, 44.4% and 31.8% with Comet compared with Megatron-Cutlass, Megatron-TE, FasterMoE and Tutel respectively. The performance gain is more prominent with the identical attention computation apart. Comet outperforms other baselines in all configurations because it realizes sufficient overlapping and the scheduling inside high-performance fused kernels greatly reduce the the overhead at CPU side.

Besides, we can also observe that Megatron-Cutlass and Megatron-TE perform similar. This is because they are identical except from the implementation of GEMM/GroupGEMM. Neither of them supports overlapping, while Megatron-TE performs worse in some cases because of the overhead in transformer engine API calls. Tutel performs better than other baselines because it incorporates communication into experts' computation through delicate scheduling and adaptive parallelism. Although communication and computation is overlapped partially, when the number of experts is large (Qwen2), the advantage of Tutel diminishes because of the large scheduling overhead. FasterMoE only supports expert parallelism ($EP = W$) and it also does not perform well on Qwen2 because the experts are small in Qwen2 and the kernel invoking time for experts dominates the MoE layer.

5.3 Detailed Evaluation on a Single MoE Layer

We then conduct an in-depth examination of a single MoE layer to perform a detailed analysis.

Handling varying input token lengths. The latency of a single MoE layer with varying input token lengths is shown in Figure 10. With the input token number varying, Comet experiences a shorter duration compared with baselines and the improvement is stable. Comet achieves a 1.28× to 2.37× speedup

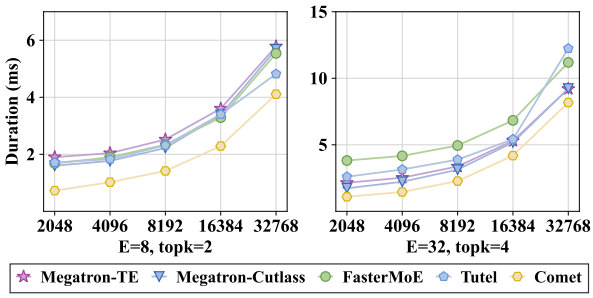

Figure 10 Single MoE layer duration with expert parallelism ($EP = 8$). The x-axis represents the total input token length M. Each device has M/W tokens before token dispatching. The shape of experts are identical to that of Mixtral 8x7B.

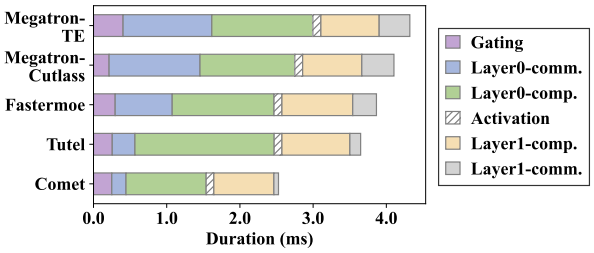

Figure 11 Time breakdown of an MoE layer with expert parallelism. ($EP = 8, TP = 1, E = 8, topk = 2$ and $M = 16384$).

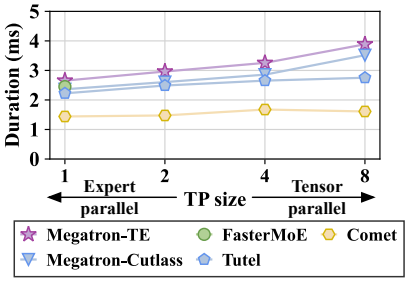

Figure 12 Single MoE layer duration under various parallelism strategies with $E = 8, topk = 2, M = 8192, EP \times TP = 8$.

compared with the baselines on average. It is noted that the advantage of COMET is prominent especially when M is small. This is because the scheduling time on the host side predominates the overall duration when M is small and COMET reduces such overhead through kernel scheduling within the fused kernel. The scheduling overhead increases with $topk$ and E for mechanisms with kernel-level scheduling (FASTERMOE and TUTEL) because the experts to manage become more complicated, inducing more kernels to be scheduled.

Time breakdown analysis of an MoE layer. The time breakdown of a specific MoE layer is shown in Figure 11. Note that the communication part only consists of the GPU-to-GPU communication time, and the operations of token indexing, dispatching and combining on local device are regarded as the computation part. As revealed, MEGATRON-TE and MEGATRON-CUTLASS experience no overlapping between communication and computation. FASTERMOE reduces the communication latency through customized Scatter and Gather operators, while the introduced local indexing extends the computation time. TUTEL reduces the communication overhead through the optimized all-to-all primitive design. However, its optimized all-to-all also exacerbates the burden of local computation. MEGATRON-TE has no communication overlapped. COMET hides 86.5% of communication latency on average and the computational efficiency of experts is not influenced, while FASTERMOE and TUTEL hide only 29.2% and 68.6% respectively.

Parallelism within the MoE layer. Because of the introduction of expert parallelism, the parallel strategy within the MoE layer can be different from the model's overall parallel strategy. Figure 12 shows the performance of methods applying diverse parallel strategies. Among all baselines, FASTERMOE unfortunately does not support tensor parallelism. For other baselines (MEGATRON-TE, MEGATRON-CUTLASS and TUTEL), the MoE layer latency increases when TP grows. This is because that tensor parallelism splits each expert onto multiple devices, triggering more fragmented small GEMMs for experts and resulting in a degradation of computational efficiency. Nevertheless, COMET maintains low latency in diverse parallelisms as the shared tensor is rescheduled to maintain computational efficiency and the weight switching overhead is eliminated.

5.4 Adaptiveness to Different Configurations

We further inquire into the performance of COMET when adapting different model configurations, runtime workloads and system environments.

Performance with various MoE parameters. We adjust the number of experts E as well as $topk$ to evaluate the performance of COMET in various MoE structures. The results are shown in Figure 13. With the increasing of $topk$, the duration of the MoE layer is increased because the computation amount at run-

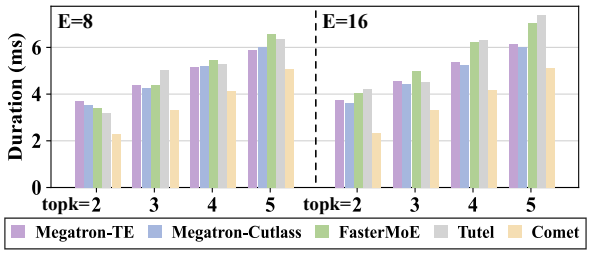

Figure 13 Duration of a single MoE layer ($M = 16384, EP = 8, TP = 1$) with various number of experts E and $topk$.

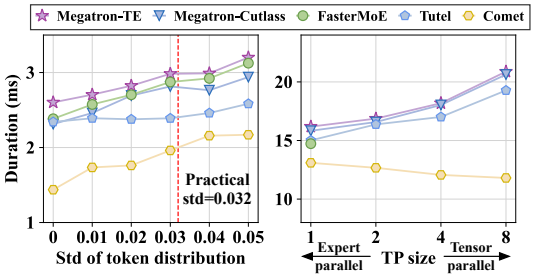

Figure 14 Performance of a MoE layer when scaling to different scenarios. Left: Duration with various token distribution with expert parallelism ($E = 8, topk = 2, M = 8192, TP = 1, EP = 8$). Right: Duration on a L20 Cluster with diverse parallelisms ($E = 8, topk = 4, M = 8192, EP \times TP = 8$).

time is scaled up. COMET consistently demonstrates superior performance across different values of $topk$ and E, yielding a speedup in the range of 1.16× to 1.83× compared to baseline implementations.

Performance with varying token distribution. When using expert parallelism, the number of tokens routed to different devices varies. We evaluate the performance of COMET in scenarios with imbalanced token distribution. The standard deviation of the token distribution across different experts is denoted as std. As shown in the left panel of Figure 14, 8192 tokens are distributed across various experts with differing distributions. When $std = 0$, tokens are uniformly distributed and each expert receives $M \times topk/E = 2048$ tokens. At $std = 0.05$, the least-loaded expert is assigned only a few hundred tokens. In a typical training job in production, the average std is 0.032. When the load imbalance problem is exacerbated, the latency of the MoE layer in all systems is prolonged. COMET consistently outperforms other MoE systems.

Scaling to distinct clusters. We carry out the experiments on another distinct cluster with a different network environment. The cluster is equipped with 8 Nvidia L20 GPUs (46 GB memory) and the GPUs are connected via PCIe bridges. The GPU-to-GPU bandwidth is around 25 GB/s as tested, which is much lower than the H800 cluster. The experiments on the L20 cluster represents a bandwidth-limited environment. As shown in the right panel of Figure 14, the average speedup of COMET compared with other baselines is from 1.19× to 1.46×. The results manifest the superiority of COMET under different cluster environments.

Table 3 Required device memory size for NVSHMEM.

Mem(MB)	Mixtral 8x7B	Qwen2-MoE	Phi3.5-MoE
M=4096	32	16	32
M=8192	64	32	64

5.5 Overhead Analysis

COMET leverages NVSHMEM to allocate a shared memory buffer for communication on each device. The buffer size is dependent on the model configuration and equals to MN, where M is the input sequence length and N is the model hidden size. For datatype of BF16 or FP16, the allocated memory size is $2MN$. The communication buffer is global for the execution of the entire model, which means that it is shared across layers and experts. We list the device memory consumption of COMET in Table 3, and it is negligible compared with the large device memory on current GPUs.

6 Related Work

With the successful application of MoE in large-scale distributed training and inference, there are plenty of works focusing on the system-level optimizations of reducing the communication overhead inherited in the MoE structure.

Communication optimization. To reduce the communication overhead in MoE execution, a straightforward approach is to leverage efficient communication algorithms [19, 30] for faster data transmission. Recent works [10, 17, 27] also propose the 2D-hierarchical all-to-all algorithm to better utilize intra-node bandwidth and accelerate MoE communication. Some other works propose to reduce communication volume by data compression. For example, ScheMoE [32] and Zhou et al., [39] propose to apply data compression technologies to reduce the all-to-all communication volume while preserving the model convergence.

Computation-communication overlapping. The techniques of overlapping of computation and communication for dense models have been extensively employed in distributed training and inference [4, 5, 11, 34, 37, 38]. For the MoE structure, recent studies also try to identify the pipelining opportunities for communication tasks of all-to-all operations and computing tasks of GEMMs. FasterMoE [8] 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. PipeMoE [31] and ScheMoE [32] aim to schedule MoE operators to better utilize intra- and inter-connect bandwidths. These solutions realize overlapping through kernel-level scheduling and do not fully resolve the fine-grained data dependency in MoE.

7 Conclusion

In this paper, we propose Comet, a MoE system that aims to achieve fine-grained communication and computation overlapping for MoE. Comet features two key designs to achieve seamless overlapping without impact the computational efficiency: Shared tensor based dependency resolving that enables fine-grained overlapping, while eliminating the bottleneck caused by fine-grained communication I/O; The workload assignment mechanism that promises precise and adaptive overlapping of operators, inducing maximal latency concealing. Comet achieves 1.96× speedup in a single MoE layer and 1.71× speedup in the end-to-end execution of MoE models, compared with existing literature.

References

[1] Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. arXiv preprint arXiv:2404.14219, 2024.

[2] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A frontier large vision-language model with versatile abilities. arXiv preprint arXiv:2308.12966, 2023.

[3] Bing Cao, Yiming Sun, Pengfei Zhu, and Qinghua Hu. Multi-modal gated mixture of local-to-global experts for dynamic image fusion. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 23555–23564, 2023.

[4] Liwen Chang, Wenlei Bao, Qi Hou, Chengquan Jiang, Ningxin Zheng, Yinmin Zhong, Xuanrun Zhang, Zuquan Song, Ziheng Jiang, Haibin Lin, et al. Flux: Fast software-based communication overlap on gpus through kernel fusion. arXiv preprint arXiv:2406.06858, 2024.

[5] Chang Chen, Xiuhong Li, Qianchao Zhu, Jiangfei Duan, Peng Sun, Xingcheng Zhang, and Chao Yang. Centauri: Enabling efficient scheduling for communication-computation overlap in large model training via communication partitioning. In Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, pages 178–191, 2024.

[6] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), pages 578–594, 2018.

[7] Jiaao He, Jiezhong Qiu, Aohan Zeng, Zhilin Yang, Jidong Zhai, and Jie Tang. Fastmoe: A fast mixture-of-expert training system. arXiv preprint arXiv:2103.13262, 2021.

[8] Jiaao He, Jidong Zhai, Tiago Antunes, Haojie Wang, Fuwen Luo, Shangfeng Shi, and Qin Li. Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models. In Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pages 120–134, 2022.

[9] Huggingface. Hugging face. https://huggingface.co/, 2022.

[10] Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, et al. Tutel: Adaptive mixture-of-experts at scale. Proceedings of Machine Learning and Systems, 5:269–287, 2023.

[11] Abhinav Jangda, Jun Huang, Guodong Liu, Amir Hossein Nodehi Sabet, Saeed Maleki, Youshan Miao, Madanlal Musuvathi, Todd Mytkowicz, and Olli Saarikivi. Breaking the computation and communication abstraction barrier in distributed machine learning workloads. In Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, pages 402–416, 2022.

[12] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. arXiv preprint arXiv:2401.04088, 2024.

[13] 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. arXiv preprint arXiv:2006.16668, 2020.

[14] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. Advances in neural information processing systems, 36, 2024.

[15] Rui Liu, Young Jin Kim, Alexandre Muzio, and Hany Hassan. Gating dropout: Communication-efficient regularization for sparsely activated transformers. In International Conference on Machine Learning, pages 13782–13792. PMLR, 2022.

[16] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In Proceedings of the IEEE/CVF international conference on computer vision, pages 10012–10022, 2021.

[17] Xiaonan Nie, Pinxue Zhao, Xupeng Miao, Tong Zhao, and Bin Cui. Hetumoe: An efficient trillion-scale mixture-of-expert distributed training system. arXiv preprint arXiv:2203.14685, 2022.

[18] Nvidia. Unified memory for cuda beginners. https://developer.nvidia.com/blog/unified-memory-cuda-beginners/, 2017.

[19] Nvidia. Doubling all2all performance with nvidia collective communication library 2.12. https://developer.nvidia.com/blog/doubling-all2all-performance-with/nvidia-collective-communication/library-2-12/, 2022.

[20] Nvidia. Nvidia hopper architecture in-depth. https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/, 2022.

[21] Nvidia. Cutlass. https://github.com/NVIDIA/cutlass, 2024.

[22] Nvidia. Grouped gemm for moe. https://github.com/fanshiqing/grouped_gemm, 2024.

[23] Nvidia. Nccl. https://developer.nvidia.com/nccl, 2024.

[24] Nvidia. Nvshmem. https://developer.nvidia.com/nvshmem, 2024.

[25] Nvidia. Transformer engine. https://github.com/NVIDIA/TransformerEngine, 2024.

[26] OpenAI. Introducing triton: Open-source gpu programming for neural networks. https://openai.com/index/triton/, 2022.

[27] 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. In International conference on machine learning, pages 18332–18346. PMLR, 2022.

[28] Or Sharir, Barak Peleg, and Yoav Shoham. The cost of training nlp models: A concise overview. arXiv preprint arXiv:2004.08900, 2020.

[29] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. arXiv preprint arXiv:1701.06538, 2017.

[30] Liang Shen, Zhihua Wu, WeiBao Gong, Hongxiang Hao, Yangfan Bai, HuaChao Wu, Xinxuan Wu, Jiang Bian, Haoyi Xiong, Dianhai Yu, et al. Se-moe: A scalable and efficient mixture-of-experts distributed training and inference system. arXiv preprint arXiv:2205.10034, 2022.

[31] Shaohuai Shi, Xinglin Pan, Xiaowen Chu, and Bo Li. Pipemoe: Accelerating mixture-of-experts through adaptive pipelining. In IEEE INFOCOM 2023-IEEE Conference on Computer Communications, pages 1–10. IEEE, 2023.

[32] Shaohuai Shi, Xinglin Pan, Qiang Wang, Chengjian Liu, Xiaozhe Ren, Zhongzhe Hu, Yu Yang, Bo Li, and Xiaowen Chu. Schemoe: An extensible mixture-of-experts distributed training system with tasks scheduling. In Proceedings of the Nineteenth European Conference on Computer Systems, pages 236–249, 2024.

[33] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. arXiv preprint arXiv:1909.08053, 2019.

[34] Jaeyong Song, Jinkyu Yim, Jaewon Jung, Hongsun Jang, Hyung-Jin Kim, Youngsok Kim, and Jinho Lee. Optimus-cc: Efficient large nlp model training with 3d parallelism aware communication compression. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, pages 560–573, 2023.

[35] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288, 2023.

[36] A Vaswani. Attention is all you need. Advances in Neural Information Processing Systems, 2017.

[37] Shibo Wang, Jinliang Wei, Amit Sabne, Andy Davis, Berkin Ilbeyi, Blake Hechtman, Dehao Chen, Karthik Srinivasa Murthy, Marcello Maggioni, Qiao Zhang, et al. Overlap communication with dependent computation via decomposition in large deep learning models. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1, pages 93–106, 2022.

[38] Yuke Wang, Boyuan Feng, Zheng Wang, Tong Geng, Kevin Barker, Ang Li, and Yufei Ding. {MGG}: Accelerating graph neural networks with {Fine-Grained}{Intra-Kernel}{Communication-Computation} pipelining on {Multi-GPU} platforms. In 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23), pages 779–795, 2023.

[39] Qinghua Zhou, Pouya Kousha, Quentin Anthony, Kawthar Shafie Khorassani, Aamir Shafi, Hari Subramoni, and Dhabaleswar K Panda. Accelerating mpi all-to-all communication with online compression on modern gpu clusters. In International Conference on High Performance Computing, pages 3–25. Springer, 2022.

[40] Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-experts with expert choice routing. Advances in Neural Information Processing Systems, 35:7103–7114, 2022.

[41] 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.