Harnessing Inter-GPU Shared Memory for Seamless MoE Communication-Computation Fusion

PPoPP'25

Hulin Wang[1], Yaqi Xia[1], Donglin Yang[2], Xiaobo Zhou[3], Dazhao Cheng[1]

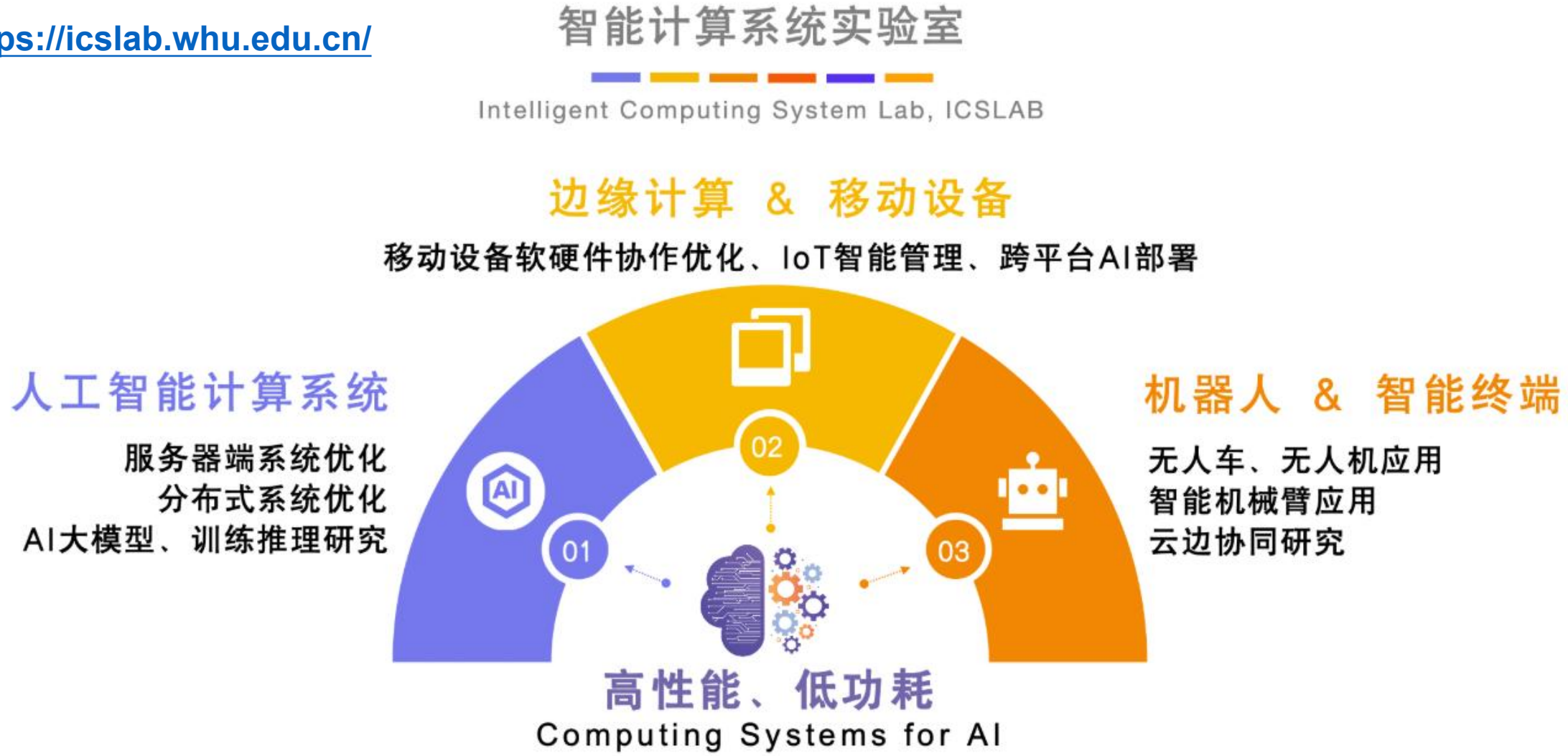
Wuhan University[1], Nvidia Corporation[2], University of Macau[3]

Presenter: 陈宇杰

2026.1.13

Author

https://icslab.whu.edu.cn/

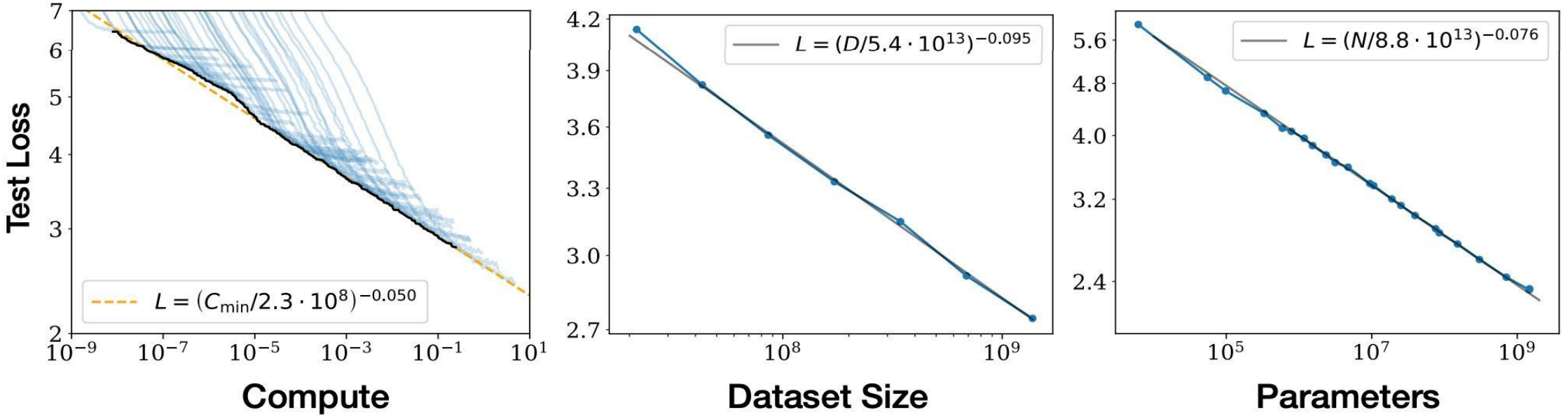


Contents

- Background
- Design
- Evaluation
- Thinking

Background

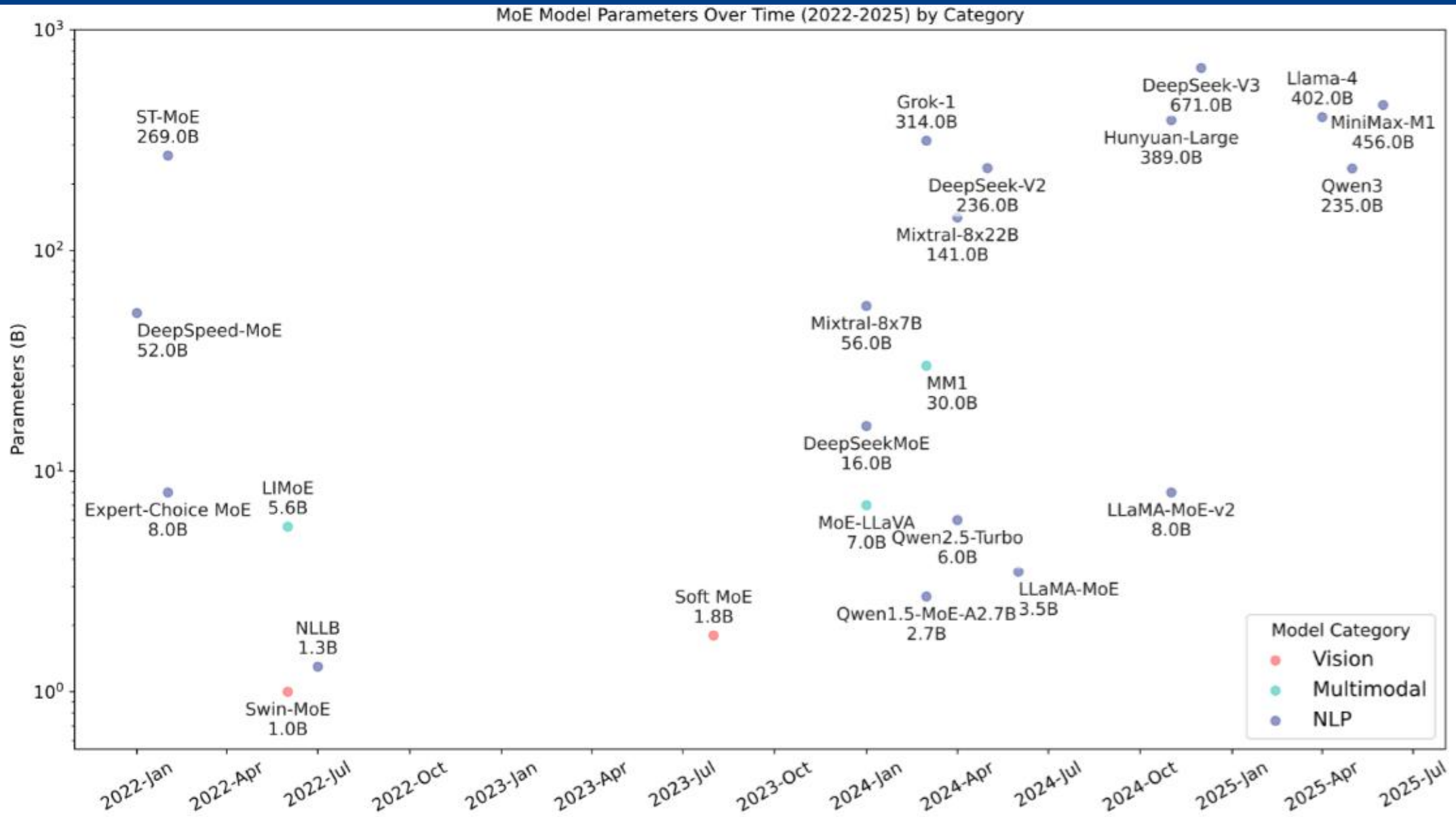
OpenAI提出的Scaling Laws指出模型的性能与计算资源、数据集大小及模型参数量之间呈现显著的幂律关系。



- 性能增益：随着模型参数、数据和算力的增加，模型性能持续提升，未出现明显的饱和现象。
- 资源瓶颈：传统稠密模型 (Dense Models) 的计算成本随参数量线性增长，导致在大规模扩展时面临巨大的算力与能耗挑战。

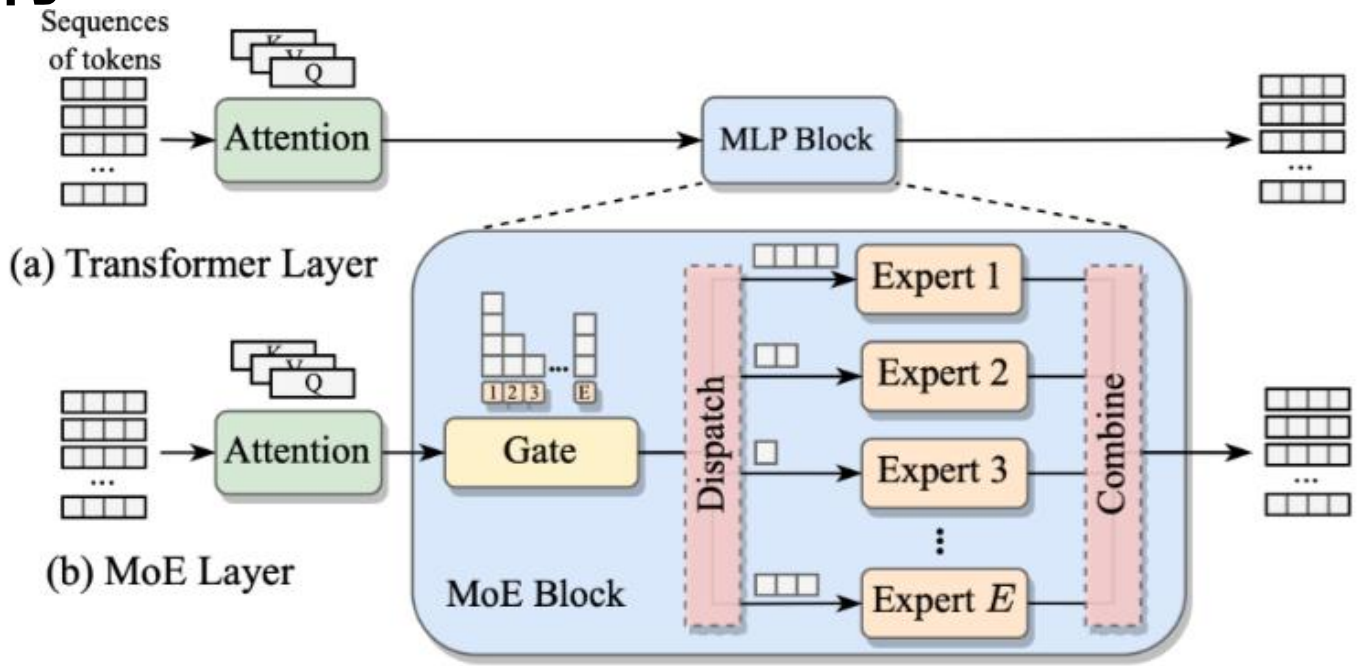
为了在**扩大参数规模**的同时**控制计算成本**，引入混合专家模型 (Mixture of Experts)。通过**条件计算**，实现“大参数量，低计算量”的高效扩展。

Background


MoE Model Parameters Over Time (2022-2025) by Category
Parameters (B)
10^3
10^2
10^1
10^0
ST-MoE
269.0B
DeepSpeed-MoE
52.0B
Expert-Choice MoE
8.0B
LIMoE
5.6B
NLLB
1.3B
Swin-MoE
1.0B
Soft MoE
1.8B
Grok-1
314.0B
DeepSeek-V2
236.0B
Mixtral-8x22B
141.0B
Mixtral-8x7B
56.0B
MM1
30.0B
DeepSeekMoE
16.0B
MoE-LLaVA
7.0B
Qwen2.5-Turbo
6.0B
LLaMA-MoE
3.5B
Qwen1.5-MoE-A2.7B
2.7B
DeepSeek-V3
671.0B
Hunyuan-Large
389.0B
Llama-4
402.0B
MiniMax-M1
456.0B
Qwen3
235.0B
LLaMA-MoE-v2
8.0B
2022-Jan
2022-Apr
2022-Jul
2022-Oct
2023-Jan
2023-Apr
2023-Jul
2023-Oct
2024-Jan
2024-Apr
2024-Jul
2024-Oct
2025-Jan
2025-Apr
2025-Jul
Model Category
Vision
Multimodal
NLP

Background

混合专家模型结构

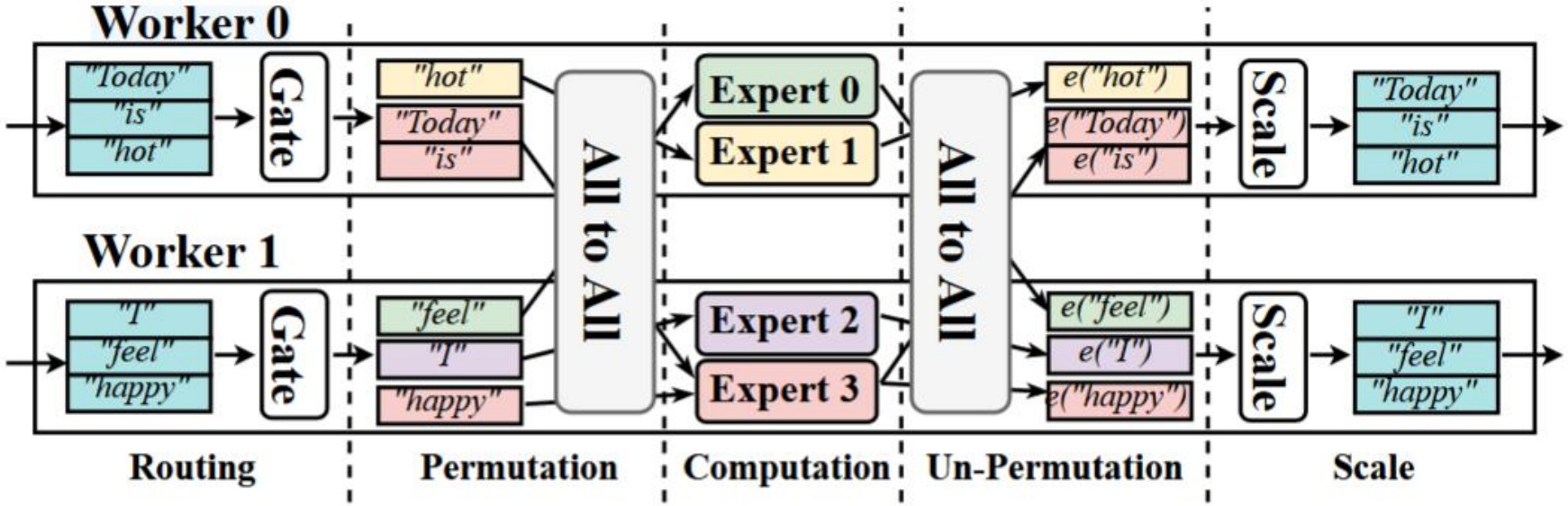


混合专家模型对Transformer的前馈层进行了稀疏化改造，包含两个核心组件：

- 门控模块 (Gating Module)：引入可学习的门控网络，负责根据输入Token的特征，将其动态路由至最匹配的一个或多个专家。
- 稀疏专家层 (Sparse MoE Layers)：原有的单一大规模前馈网络 (FFN) 被替换为多个并行的“专家”网络。虽然模型总参数量巨大，但对于每个特定输入，仅有极少部分模型参数被激活。

Background

MoE架构中的专家并行与通信机制



为了突破单个计算节点显存容量对模型总参数规模的限制，使用**专家并行**将不同的专家模型 (Experts) 物理部署在多个计算节点上，而输入Token则根据路由结果在节点间传输。

- 前向分发:门控网络计算路由后，通过All-to-All通信原语，将位于不同工作节点上的Token发送至其对应的目标专家所在的节点。
- 结果聚合:专家完成计算后，再次通过All-to-All通信，将计算结果传回Token原本所在的节点进行后续处理。

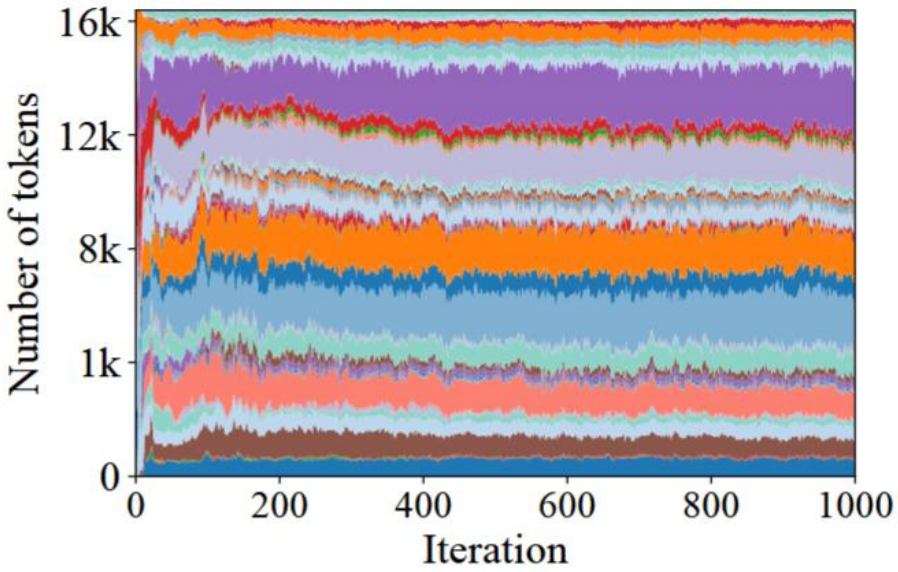
All-to-All操作通常是阻塞的，即计算单元必须等待通信完成后才能继续执行，引入了同步屏障。

Background

MoE架构下的挑战：通信瓶颈与负载不均衡

Table 1: Time consumption within a MoE layer.

# Layers	Batch size	# Experts	A2A Ratio
12	16	16	56.44 %
24	16	16	57.57 %
24	32	16	57.20 %
12	16	32	56.13 %
12	16	64	56.34 %



1. 高昂的通信开销

- All-to-All通信与数据同步时间占据总训练时长的50%以上，随着训练规模的加大，通信延迟成为制约模型训练效率的首要瓶颈。
- 同步阻塞：在通信期间，计算单元处于空闲等待状态，导致硬件利用率低。

2. 专家激活的偏斜

- 专家负载倾斜：门控网络（Gate）导致Token分布极度不均，热门专家处理大量数据，冷门专家空闲。
- 木桶效应：分布式训练速度受限于最慢的（负载最重的）GPU，且跨节点传输加剧了通信压力。

Background-related work

现有负载均衡策略和局限

背景问题：动态路由导致Token分布不均，引发GPU间的负载失衡。

- **Token容量限制**

策略：限制每个专家处理的Token数量（Fastermoe PPoPP'22）。

缺陷：导致部分Token被重路由至次优专家或直接丢弃，损害模型质量。

- **专家重分配 (Expert Reassignment)**

策略：动态调整冷热专家在GPU上的分布以平衡负载（Flexmoe SIGMOD'23）。

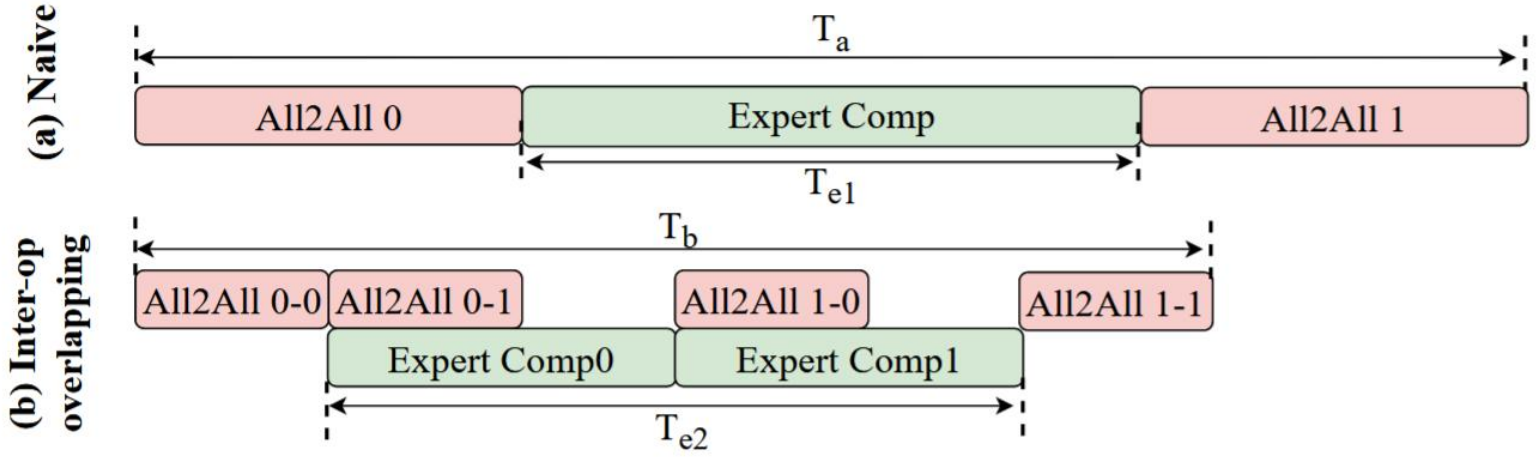
缺陷：引入了额外的专家迁移通信开销，且重分配算法主要占用通信资源，导致硬件利用率低。

需要一种既能保持模型质量，又能避免额外通信开销的高效负载均衡策略。

Background-related work

现有的通信优化及局限

核心思想：通信基于NVIDIA NCCL (All-to-All)，将输入数据切分为小批次 (Micro-batches)，利用调度算法实现通信与计算的重叠。



现有方法局限:

- 痛点 1：无法完全掩盖，依赖NCCL库进行粗粒度通信，在流水线的初期和末期，GPU仍不可避免地处于空闲状态。
- 痛点 2：被动通信，通信必须由CPU发起，数据切分引入了额外的Kernel启动开销，无法在GPU内部灵活地进行细粒度的通信/计算编排。
- 痛点 3：计算效率下降，数据切得越碎，GEMM的计算块就越小，导致计算强度降低，无法跑满GPU性能 。

Background

NVSHMEM (NVIDIA Shared Memory Library)

1. 统一地址空间 (Partitioned Global Address Space):

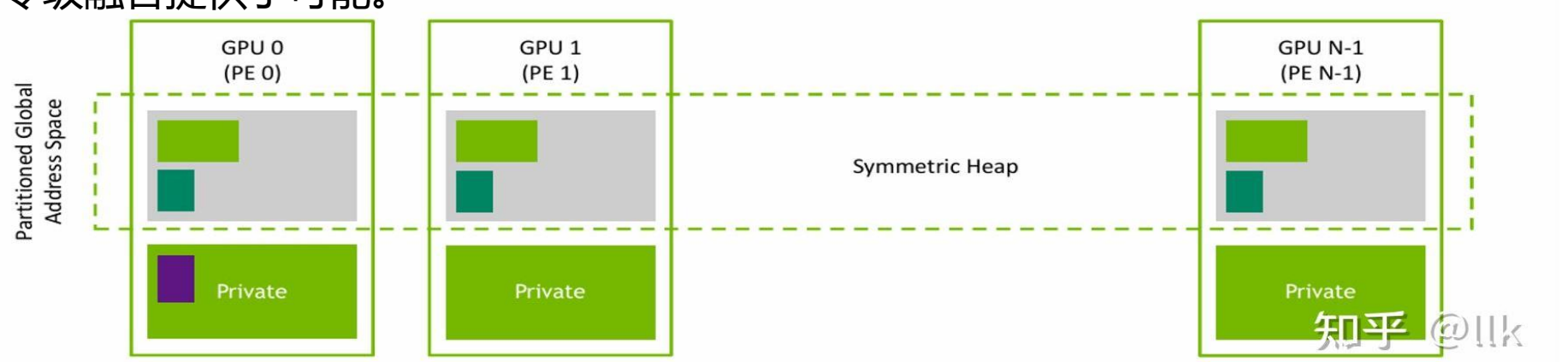
创建一个跨多个GPU内存的全局地址空间，将显存划分为“私有区”与“共享窗口”。支持跨节点零拷贝(Zero-Copy) 访问，可直接通过指针读写其它GPU的数据，绕过复杂的显存拷贝流程。

2. 单边通信 (One-sided Communication):

核心优势在于GPU间的直接通信，支持Put/Get语义，通信由发起方单方面完成，无需接收方参与同步，彻底消除了NCCL中CPU握手与全局同步的开销。

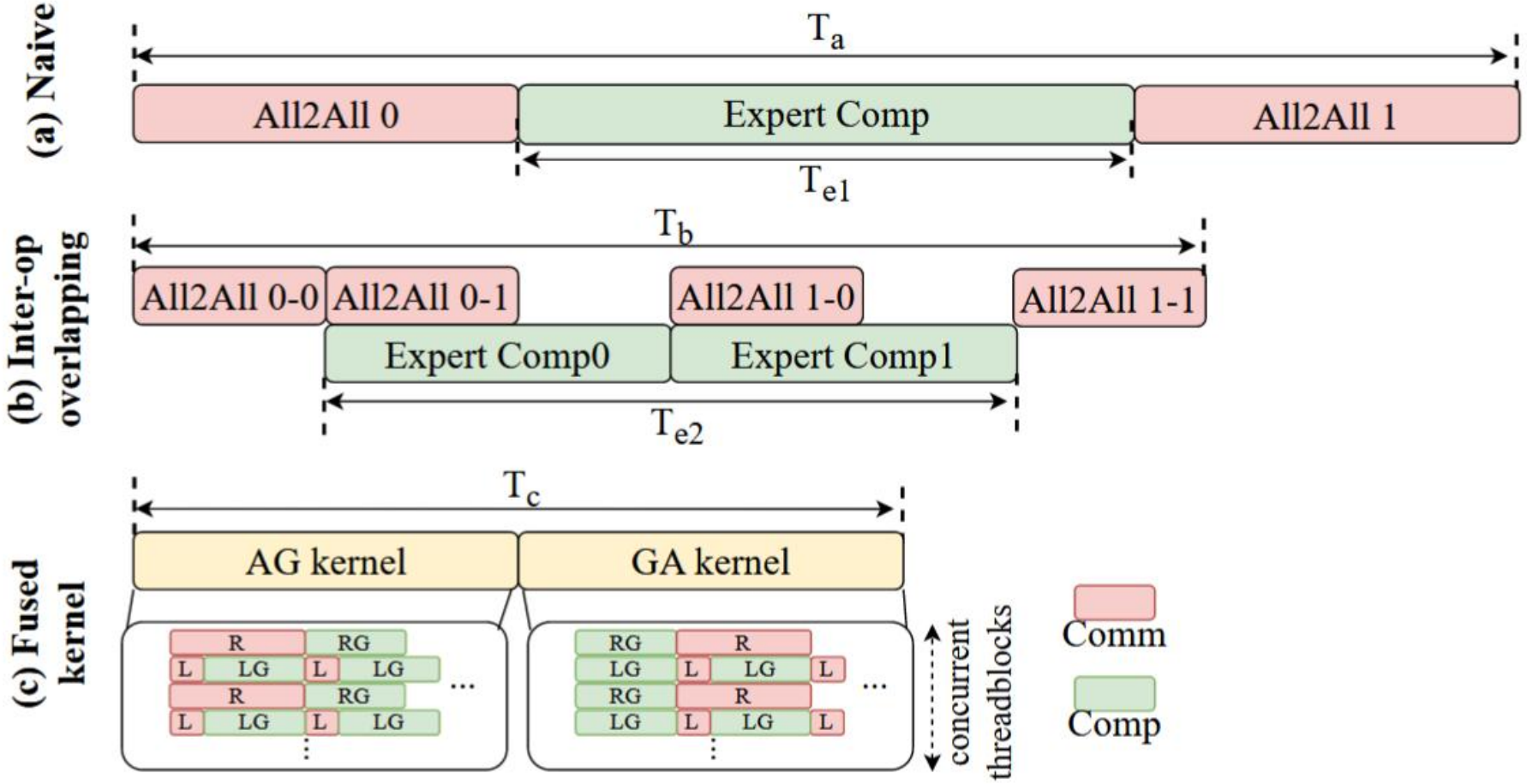
3. Kernel内发起 (Kernel-initiated):

支持Kernel内部发起细粒度通信。支持并发的CUDA Threads在计算本地数据的同时请求远端数据，为实现计算与通信的指令级融合提供了可能。



Background

Enhancing MoE Efficiency via NVSHMEM:



机遇: 利用NVSHMEM的细粒度特性，通信不再是独立的阶段，而是变成了计算指令的一部分，在Kernel内部融合通信与计算，进一步减少GPU闲置。

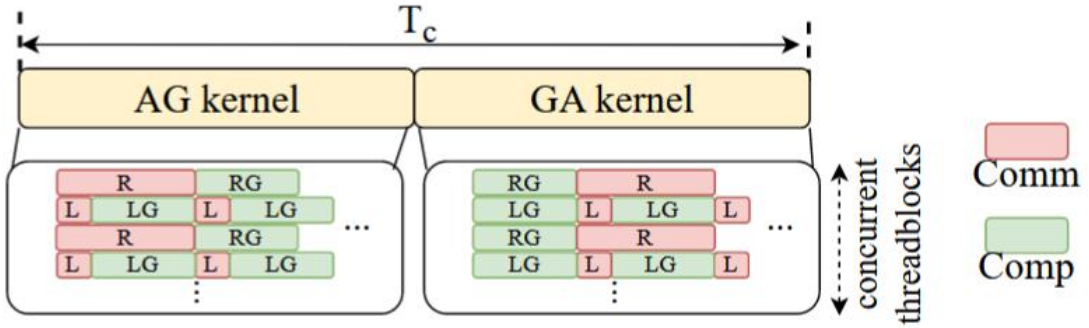
Design

CCFuser：基于NVSHMEM的计算通信无缝融合架构

解决通信瓶颈

1. 基于跨GPU共享内存的融合GEMM

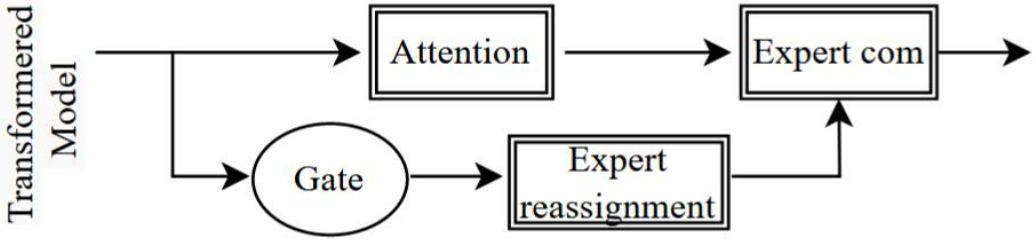
- 摒弃显式的All-to-All同步，利用NVSHMEM实现细粒度的Fused Kernel。
- 在Kernel内部同时处理本地数据并获取远端数据，实现层级化的通信掩盖。



解决负载不均

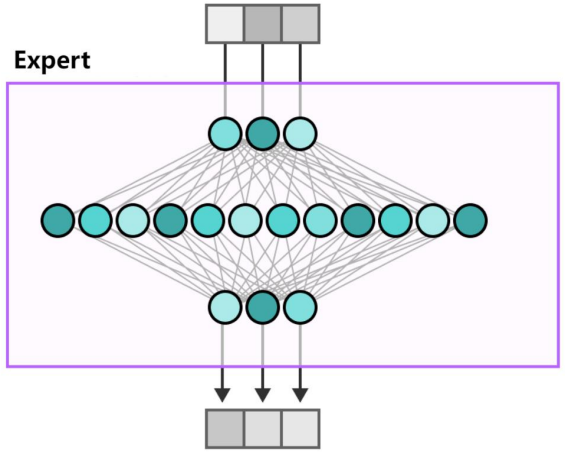
2. 基于专家热度的重分配

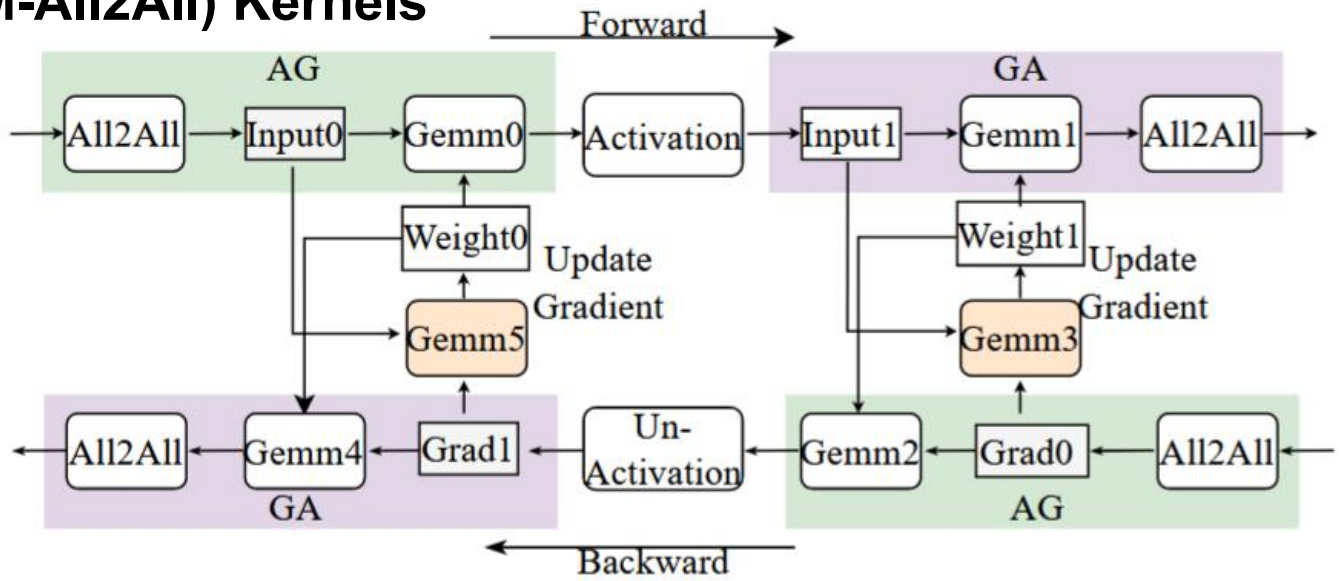
- 策略：基于专家热度动态调整专家在GPU上的分布 。
  - 热门专家: 复制以增强数据本地化。
  - 冷门专家: 按带宽拓扑重分配至空闲节点。
- 最大化数据局部性 (Data Locality)，在不牺牲模型精度的前提下平衡负载。



Design

AG (All2All-GEMM) & GA (GEMM-All2All) Kernels





利用NVSHMEM替代NCCL All2All，打破了传统 MoE 中“通信-计算”串行执行的依赖。

AG Kernel (All2All-GEMM)

- 将Input的分发 (Dispatching/Scatter) 与第一层MLP的计算彻底合并。
- 消除了Input数据就绪前的全局同步等待。

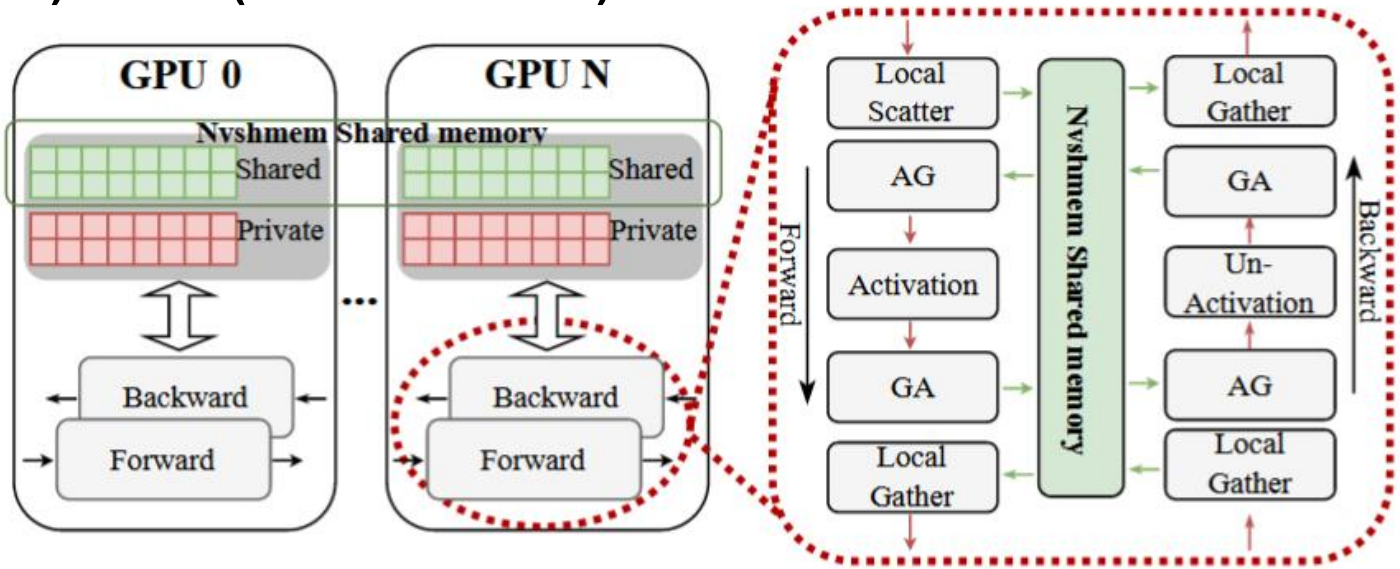
GA Kernel (GEMM-All2All)

- 将第二层 MLP 的计算与结果的聚合 (Combine/Gather) 彻底合并。
- 消除了Output结果写回前的中间缓存开销。

架构影响：通过将通信算子融合进计算内核，消除了导致GPU空转的调度边界。

Design

AG (All2All-GEMM) & GA (GEMM-All2All) Kernels



集群全局显存划分:

- Shared Segment: 存放MoE层的输入和输出，所有GPU可见，允许直接跨节点读写。
- Private Segment: 存放模型权重和激活值，仅本地可见，保护模型参数安全。

基于 NVSHMEM 的数据流:

- AG Phase(Pull Mode)：Kernel利用nvshmem_get主动从远端Shared Memory拉取输入数据。
- GA Phase(Push Mode)：Kernel计算完成后，利用nvshmem_put直接将结果写回到远端目标地址。

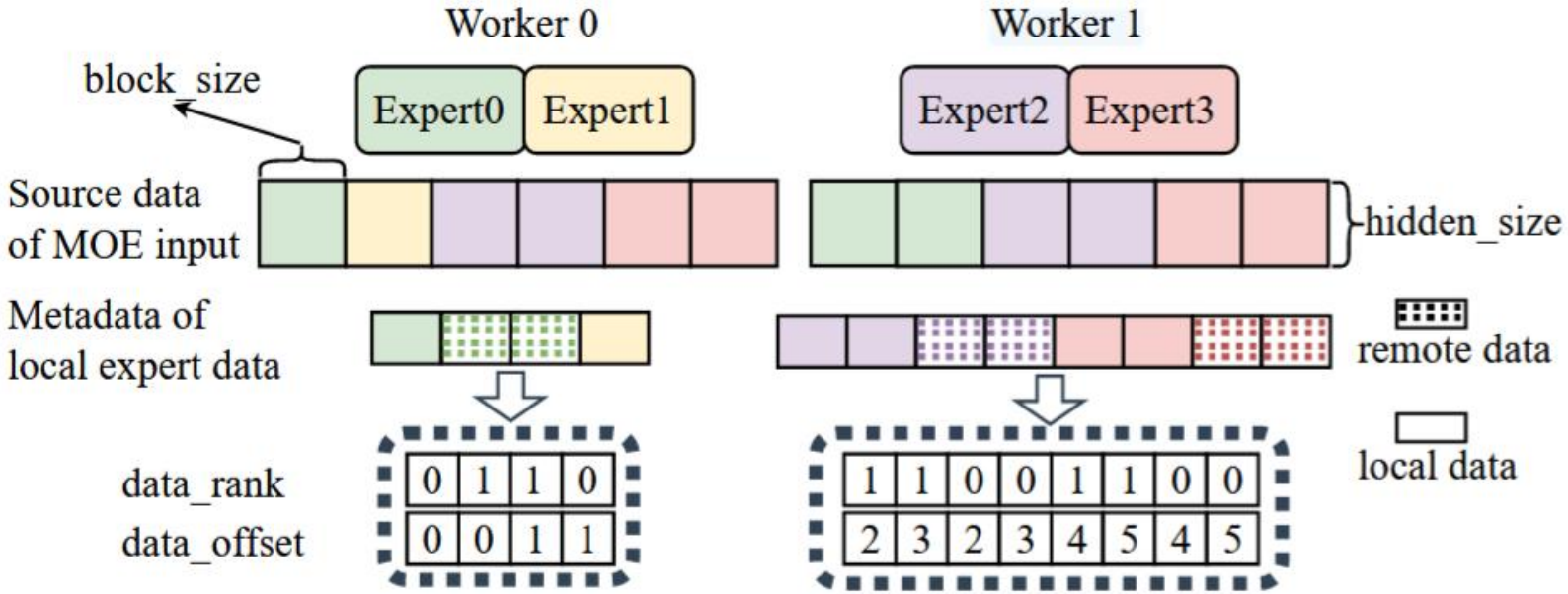
将GPU上所有专家的计算任务水平融合为一个统一的CUDA Kernel统一调度，内部GPU调度器动态分配Thread Blocks。一部分Thread Block计算本地数据，一部分处理远端数据，实现天然的并行重叠。

Design

应对动态路由 (Addressing Dynamic Routing):

挑战：MoE路由是动态的，导致Input Token在每一轮迭代中的位置都不固定，需要保证Kernel可以准确找到数据位置。

- 生成轻量级元数据Metadata，代表了当前分布在各个GPU显存里的Token，并引入data_rank（目标GPU ID）和data_offset（显存偏移量）记录远程数据在集群中的具体位置。
- Kernel启动时先读取Metadata，精确定位远端Shared Memory中的数据块。由于Metadata数据极小，因此额外的查表开销几乎可以忽略不计。

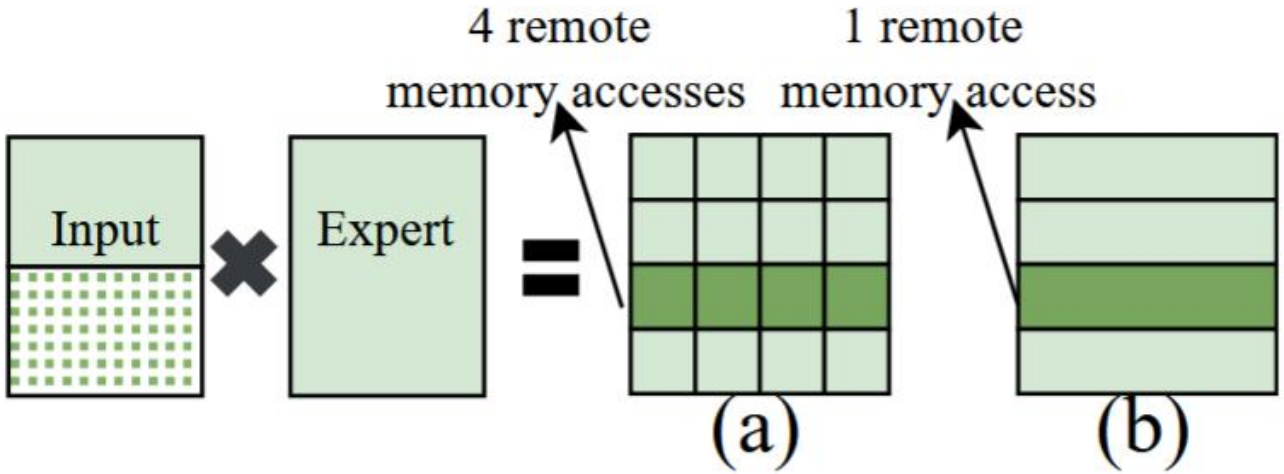


Design

优化远程访存 (Optimizing Remote Access):

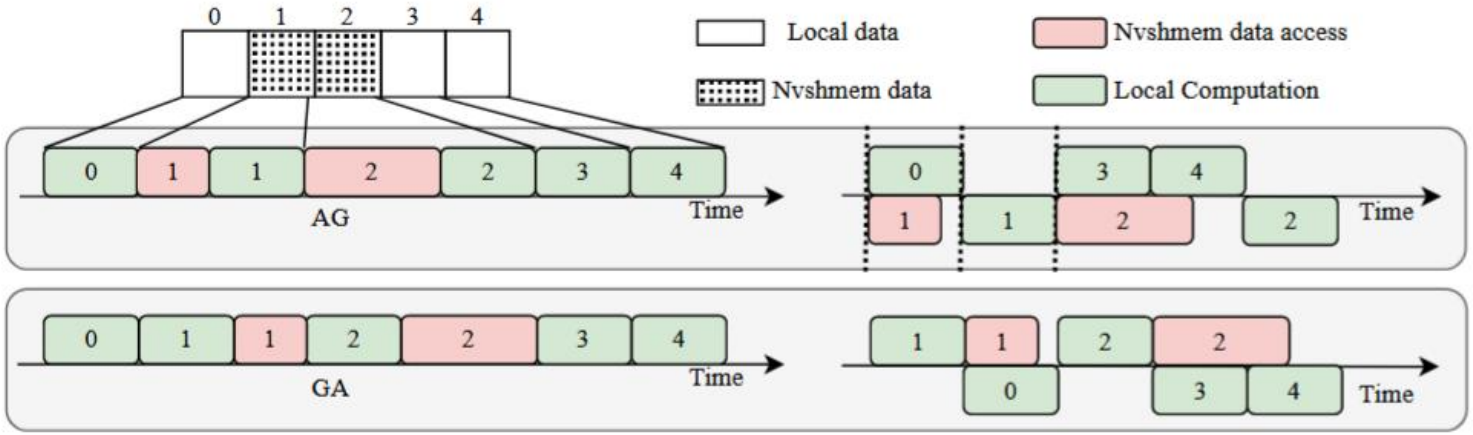
挑战：传统GEMM将矩阵切分为二维“小方块”进行并行计算，意味着计算同一行的不同列时，会有多个不同的 Threadblock 需要访问同一份输入数据，但在NVSHMEM场景下，这会导致同一块远端数据被多个Thread block重复通过低带宽链路读取。

- 强制仅沿行维度切分计算任务，改变了Thread block的数据映射方式。
- 确保每个远端数据块在整个Kernel生命周期内，只通过互联链路传输一次，消除了冗余通信，将访存模式从延迟受限 (Latency-bound) 转化为带宽高效 (Bandwidth-efficient) 。



Design

细粒度流水线重叠 (Fine-grained Overlapping)



策略一：异步流水线 (Async Pipeline)

方案：建立Async Copy (DMA) + Compute流水线，将通信延迟隐藏在本地计算的时间窗口内。

- 利用NVSHMEM的异步API启动数据搬运。
- 当DMA引擎在后台搬运远程数据时，Tensor Cores全速计算本地数据。

策略二：自适应重叠 (Adaptive Overlapping)

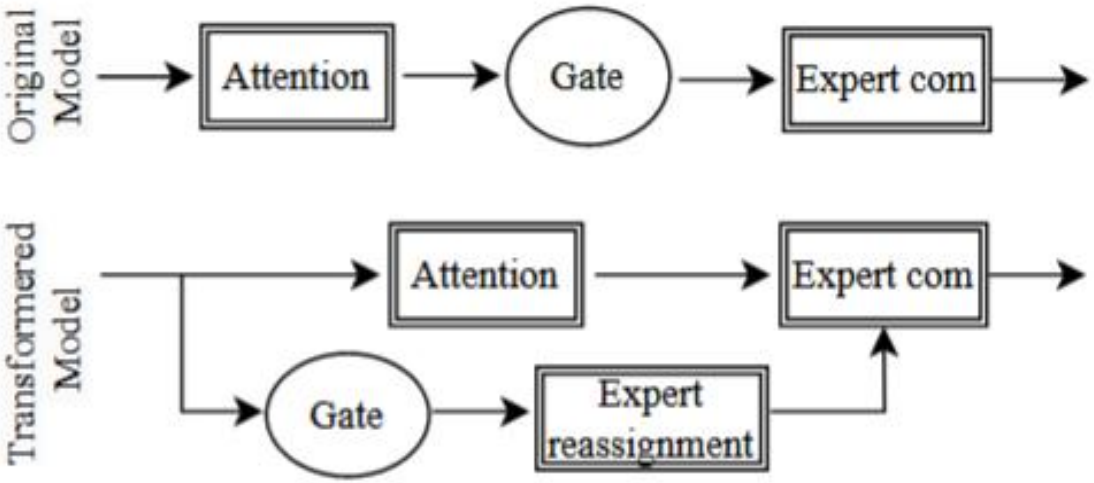
感知拓扑的动态调度：根据互联带宽（NVLink vs PCIe）自动调节本地任务量。

- NVLink: 传输快，处理1块本地数据即可掩盖延迟 (Overlap ratio 1:1)。
- PCIe: 传输慢，需要处理2-3块本地数据，利用更长的计算时间来掩盖传输 (Overlap ratio 2:1 or 3:1) 。

Design-基于热度的专家重分配策略

”零开销“的专家重分配策略

- 痛点：专家重分配算法需要计算新的布局并迁移权重。如果串行执行，会阻塞整个训练流程。
- 创新：预运行门控（Pre-run Gating），利用 Attention 层的计算时间执行专家迁移。



架构变革:解耦依赖关系，将Gate Network提前计算 。

并行流水线 (Parallel Pipeline):

- Attention层：计算密集型 (Compute-bound)，占用 GPU 算力。
- 专家重分配模块：通信密集型 (Comm-bound)，占用互联带宽。

利用Attention的计算时间，完全掩盖掉专家重分配的开销 。

Gate网络架构简单，路由决策主要依赖Token固有的语法（如名词/动词）或语义（如时间/数字）特征，这些特征对层级位置不敏感。

Design-基于热度的专家重分配策略

基于热度的分配策略：

优化目标：最大化Data Locality (数据局部性)，即让Token尽可能在本地 GPU 处理，借此掩盖远程获取数据的延迟：

$$\mathcal{L}(g', \mathcal{P}) = \frac{\sum_{e}^{(e,g')\in\mathcal{P}} \mathcal{N}_{e,g'}}{\sum_{e}^{e\in\mathcal{E}} \mathcal{N}_{e,g'}}$$

- **策略 1: 热门专家 (Popular Experts):**

  每个GPU按专家负载降序排列，选取累计Token占比超过50%的头部专家集合（确保Kernel处理本地数据的时间大于或等于拉取远端数据的时间）强制**保留或复制一份在GPU本地，**确保绝大多数Token实现本地计算，无需跨节点传输。

- **策略 2: 冷门专家 (Unpopular Experts):**

  针对负载较低的长尾专家集合G，使用基于带宽拓扑感知的迁移方式，将专家模型迁移至其他低负载GPU，且优先**选择高带宽连接**的路径，以最小化迁移带来的通信延迟。

$$g' = \arg\max_{g'} \{ \sum_{g}^{g\in\mathcal{G}} \mathcal{N}_{e',g} \times \mathcal{B}_{g,g'} \}, where\ g' \in G$$

Evaluation

硬件平台：

- 集群A (Pinnacle - PCIe)：4x NVIDIA A100 PCIe (40GB)。混合互联架构 (NVLink Bridges + PCIe Buses)
- 集群B (Vertex - NVLink)：4x NVIDIA A100 (40GB)。全互联拓扑 (NVLink)，提供600 GB/s的GPU间通信带宽

Baseline：

- FastMoE: 基于PyTorch的通用高性能MoE训练系统，作为基础性能参照
- FasterMoE: FastMoE 的改进版，通过粗粒度的通信-计算重叠调度来优化性能

评测指标 (Metrics):

- MoE Layer Time: 单个 MoE 层完成计算与通信的总耗时
- Training Step Time: 完成一次端到端训练迭代的时间
- Kernel Execution Time: 融合算子 (Fused Kernel AG/GA) 的纯执行时间

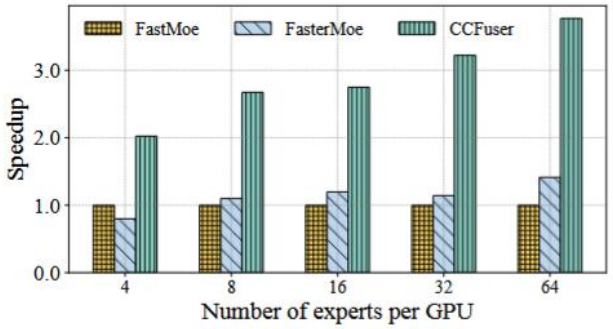
model	Batch size	Seq len	Model dim	Moe block	Total block	Expert num
M-GPT	8	1024	768	1	12	64
M-BERT	32	512	768	4	12	64
M-Trans-xl	16	512	512	12	12	64

Evaluation

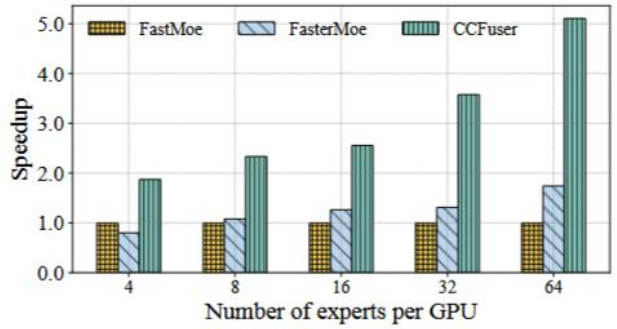
MoE层级性能

在不同专家数量 (4-64) 和两种硬件平台上进行测试

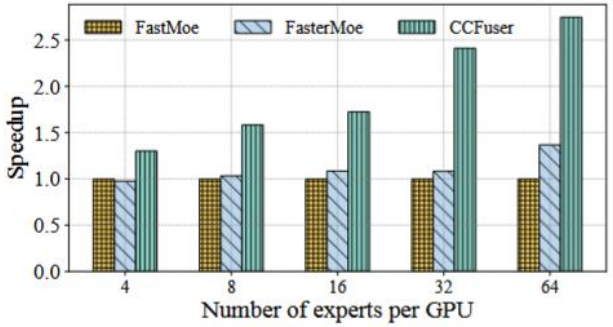
- 随着专家数量增加，加速比显著提升。相比 FastMoE，平均加速2.96x和2.0x，最高可达4.34x。
- 在PCIe平台上的提升比NVLink平台更明显。这是因为NVLink的高带宽掩盖了部分通信瓶颈，CCFuser 在通信受限环境下优势更突出。



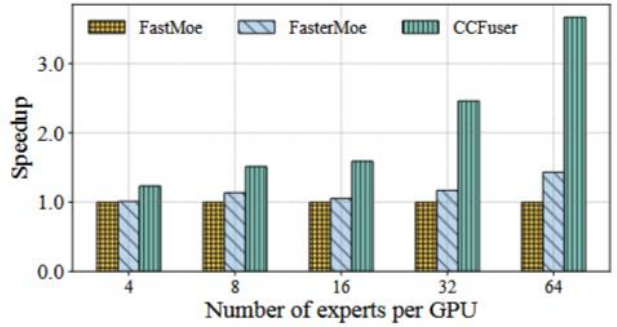
(a) Speedup of Moe layer's forward, *pinnacle.*



(b) Speedup of Moe layer's backward, *pinnacle.*



(d) Speedup of Moe layer's forward, *vertex*



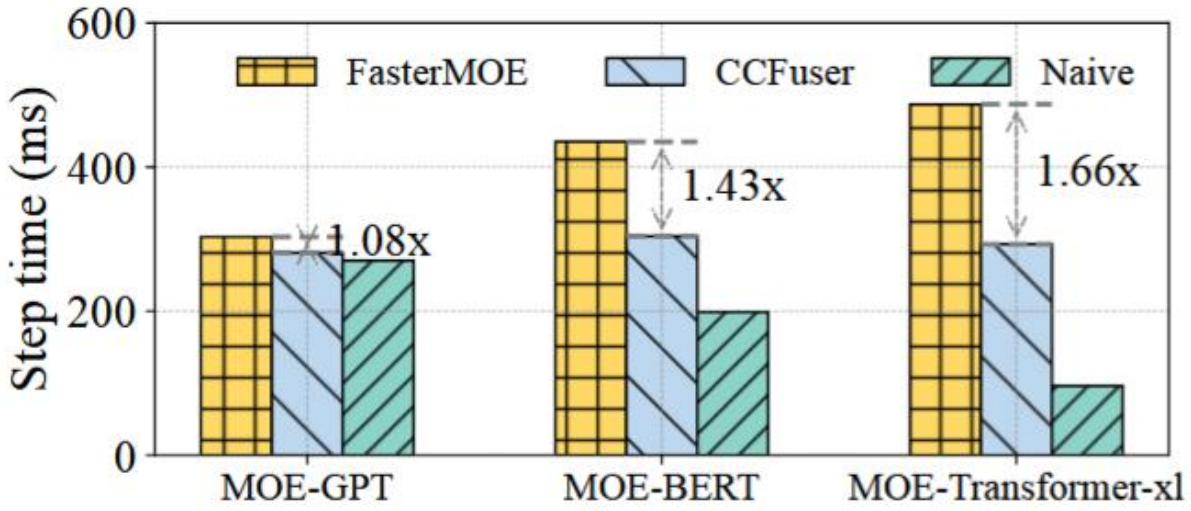
(e) Speedup of Moe layer's backward, *vertex*

Evaluation

端到端模型训练性能 (End-to-End Performance)：

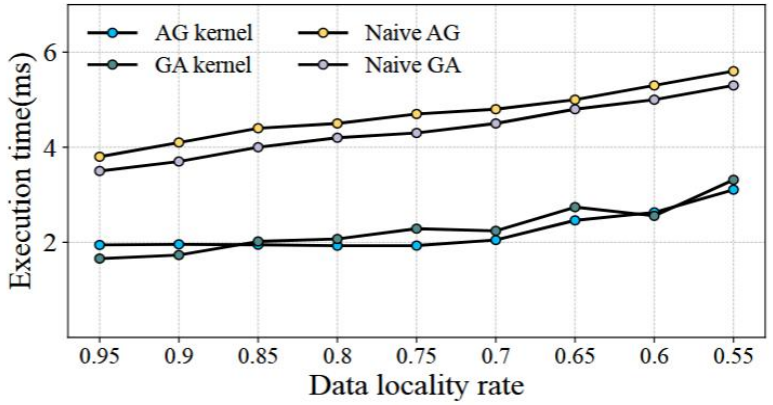
- 相比 FasterMoE，CCFuser 实现了 1.08x - 1.66x 的端到端加速。
- 端到端加速比 < MoE 层加速比。

原因: 训练步骤包含非 MoE 组件 (如 Attention, Embedding)，这些部分的开销未被 CCFuser 优化，稀释了整体加速效果。

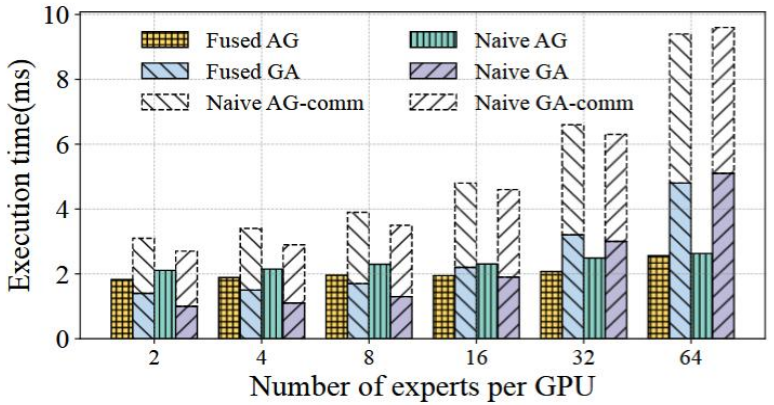


Evaluation

融合算子深度分析 (Analysis of Fused Kernel)：



(a) Varying data locality.



(b) Varying numbers of experts.

对数据局部性的鲁棒性 (Robustness to Data Locality):

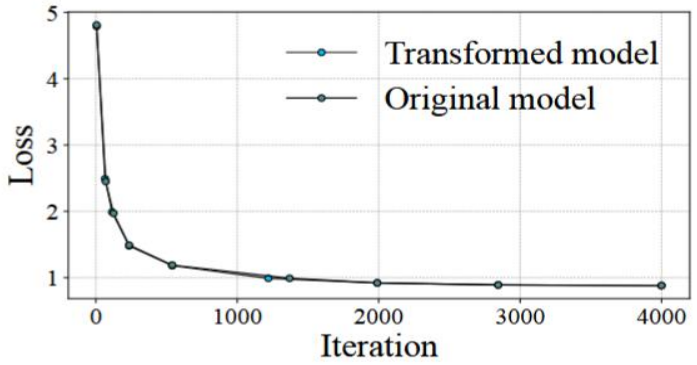
- Naive 方法: 随着数据局部性下降，通信开销激增，时间成本线性上升。
- CCFuser: 即使局部性降至 0.7 以下，性能依然保持稳定。这归功于线程块内重叠 (Intra-threadblock overlapping) 机制有效掩盖了远程访问延迟。

对专家规模的可扩展性 (Scalability with Experts)

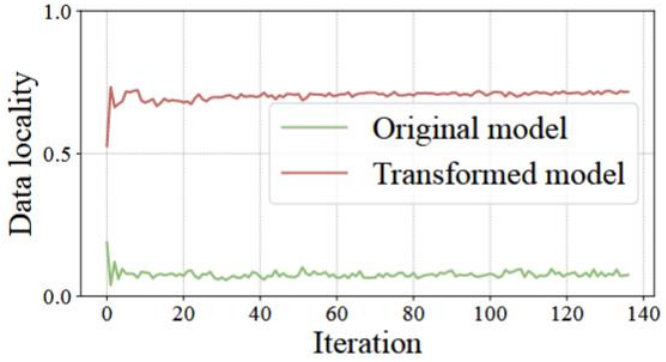
- Naive 方法: 专家越多，Kernel 启动次数越多，碎片化导致效率下降。
- CCFuser: 融合内核保持稳定的性能。

Evaluation

专家重分配策略验证 (Expert Reassignment Analysis)



(a) Loss change.



(b) Data locality change.

无损收敛 (Lossless Convergence)

- 变换后的模型与原始模型的Loss曲线基本重合。
- 并行化的Gate网络依然能准确捕捉Token路由信息，几乎不影响模型精度。

局部性显著提升 (Locality Boost)

- 效果: 本地数据局部性 (Data Locality) 从初始的 <20% 迅速提升并稳定在 ~70%。
- 收益: 高局部性大幅减少了跨节点通信需求，为 Fused Kernel提供了更优的执行环境。

Thinking

一、能不能用到我们的场景?

问题：超节点架构下的混合通信瓶颈。论文未深入探讨大规模集群中跨节点（通过PCIe/Ethernet/InfiniBand）的性能衰减问题，在万卡集群中，物理拓扑是分层的：节点内/机框内是极高带宽的 NVLink/NVSwitch，而节点间是相对低带宽的IB。CCFuser依赖NVSHMEM进行细粒度的P2P访问，但在万卡规模下，全局建立P2P连接非常困难（显存映射表会爆炸，且跨非NVLink域的P2P性能较差）。

思考：

- **分层通信域设计：**不是在全集群用 NVSHMEM。将 CCFuser 的 Fused Kernel限制在Supernode内部，在Supernode之间，仍然使用粗粒度的NCCL或定制的流水线通信。
- **超节点感知的专家调度：**确保热门专家及其副本被均匀地Scatter到不同的Supernode中，在Supernode内部，利用CCFuser的策略将该专家复制到本地显存，将大量的All-to-All流量转化为Supernode内部的高速NVSHMEM流量，只让少量的冷门专家流量走跨节点网络。

Thinking

二、问题可以泛化吗?

问题：在张量并行训练中，模型切分导致单卡仅持有部分数据，必须高频中断计算进行全量同步。传统NCCL模式受限于CPU调度开销与“计算—等待—通信”的串行机制，导致显著的流水线气泡，造成GPU算力的大量空转。

丝毫：采用“NVSHMEM + NCCL”分层治理策略。利用NVSHMEM在Kernel内部进行细粒度通信，在计算单元处理当前分块时，通信单元异步预取邻居节点的切片数据。这种流水线设计将物理通信延迟完全掩盖在计算时间内，NCCL负责宏观、粗粒度的数据同步，如全局梯度规约。

Thinking

三、能不能进一步提高?

问题：非MoE层（如Attention）成为新的性能瓶颈。 论文实验显示，尽管MoE层的加速比很高（最高4.34x），但端到端（End-to-End）的整体训练加速比受限（约1.08x-1.66x） 。主要原因是Attention模块和Embedding层未被优化，随着MoE通信开销的消除，这些部分成为新的短板 。

思考：将CCFuser的通信-计算融合思想扩展到Attention层。利用NVSHMEM来优化序列并行（Sequence Parallelism）中的通信。

Q & A

Presenter: Yujie Chen

2026.1.13