大模型训练的混合并行技术综述

张贵鹏 孙毓忠
（中国科学院计算技术研究所 北京 100190）
（中国科学院大学 北京 100049）

摘 要 在生成式人工智能的迅猛发展推动下，基于 Transformer 架构的大规模预训练模型呈现出参数规模的指数级增长。面对数百亿甚至千亿级参数模型的训练需求，传统的单模态并行方法在计算效率、内存占用和通信开销等方面面临严峻挑战，从而促使混合并行技术逐渐成为大规模分布式训练的主流范式。本文以 Transformer 架构的并行化特性为研究切入点，系统地分析了数据并行、张量并行、序列并行、流水线并行及专家并行的内在机制，揭示了不同并行策略之间的耦合关系与组合边界。通过整合算子内切分与算子间切分的数学模型，构建了混合并行策略的统一表示框架。该框架通过分离算子切分逻辑与并行拓扑映射，为现有研究提供了可扩展的理论分析工具。在方法论层面，基于该混合并行框架的理论推导，本文总结了基于计算图分解的自动并行搜索技术的发展路径。最后，结合当前技术瓶颈与新兴硬件架构，本文展望了从多模态计算协同与异构集群调度等方面的未来发展方向，为突破万亿参数模型训练的系统性挑战提供理论依据。

关键词 大模型；混合并行；自动并行；异构计算；Transformer
中图法分类号 TP316 **DOI 号** 10.11897/SP.J.1016.2026.00109

A Survey on Hybrid Parallelism Techniques for Large Model Training

ZHANG Gui-Peng SUN Yu-Zhong
(Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)
(University of Chinese Academy of Sciences, Beijing 100049)

Abstract Driven by the precipitous advancement of generative artificial intelligence, large-scale pre-trained models, particularly those based on the Transformer architecture, are exhibiting an exponential increase in parameter scale. As the community ventures into training models with hundreds of billions or even trillions of parameters, conventional uni-dimensional parallelism methods encounter formidable challenges. These traditional approaches are increasingly constrained by limitations in computational efficiency, prohibitive memory footprints on individual devices, and excessive communication overhead across the hardware cluster. Consequently, hybrid parallelism, which strategically combines multiple parallelization techniques, has emerged as the dominant and indispensable paradigm for large-scale distributed training in contemporary deep learning. This paper presents a systematic investigation into the parallelization characteristics inherent to the Transformer architecture, which serves as our primary research focus. We conduct an in-depth analysis of the foundational mechanisms underpinning five key parallelism strategies: data parallelism, tensor parallelism, sequence parallelism, pipeline parallelism, and expert parallelism. Our examination goes beyond individual techniques to meticulously uncover the intricate coupling relationships and delineate the combination boundaries that exist between these distinct

收稿日期：2025-04-09；在线发布日期：2025-11-03。本课题得到科技创新 2030 重点研发课题“众核多擎弹性加速分布式训练”(No. 2022ZD0119104)资助。**张贵鹏**，博士研究生，中国计算机学会(CCF)会员，主要研究领域为分布式训练、异构计算。E-mail：zhangguipeng23z@ict.ac.cn。**孙毓忠(通信作者)**，博士，研究员，中国计算机学会(CCF)会员，主要研究领域为云计算、操作系统。E-mail：yuzhongsun@ict.ac.cn。

strategies. This analysis is crucial for understanding how different parallelism dimensions can be synergistically composed or where they might conflict, thereby informing the design of effective and scalable training systems. A central contribution of this work is the development of a unified representation framework for hybrid parallelism strategies. This framework is constructed by integrating the mathematical models of both intra-operator splitting and inter-operator splitting. A key innovation of our framework is the explicit separation of the operator-splitting logic from the parallel topology mapping. This decoupling provides a flexible and extensible theoretical tool for analyzing existing research and designing novel hybrid configurations, as it allows for independent reasoning about how a model is parallelized and how that parallel structure is mapped onto physical hardware. On a methodological level, we leverage the theoretical derivations from our unified framework to summarize and chart the evolutionary trajectory of automatic parallelism search techniques. These techniques, which are typically based on computation graph decomposition, aim to discover optimal hybrid parallelism strategies automatically, thus alleviating the significant engineering burden of manual configuration. Our framework provides a structured lens through which the progress and remaining challenges in this domain can be systematically understood. Finally, looking ahead, we cast a forward-looking perspective on future research directions, taking into account current technological bottlenecks and the advent of emerging hardware architectures. We identify multi-modal computational collaboration and advanced scheduling for heterogeneous clusters as two of the most critical frontiers. Addressing these areas will be paramount for enabling efficient co-training of models that process diverse data types and for effectively utilizing complex computing systems with varied hardware accelerators. This paper aims to provide a robust theoretical foundation to support these future endeavors, ultimately contributing to the systemic breakthroughs required to conquer the challenges of training trillion-parameter models and beyond.

Keywords large model; hybrid parallelism; automatic parallelism; heterogeneous computing; Transformer

1 引 言

深度学习(Deep Learning)[1]是一种基于人工神经网络的机器学习方法,近年来因其在图像识别、语音识别、自然语言处理等领域取得的显著成果而备受关注。自 2012 年 Hinton 等人提出的深度卷积神经网络[2]在 ImageNet 竞赛中取得突破性成绩以来,深度学习技术迅速发展,成为人工智能研究的核心方向之一。

自 ChatGPT[3]以 1750 亿参数[4]规模引爆生成式人工智能浪潮以来,基于 Transformer 架构[5]的千亿级大语言模型迅速成为全球科技竞争的战略高地。其自注意力机制和并行处理能力使其在大规模数据和复杂任务中表现出色,不仅在自然语言处理领域取得了巨大成功,还被广泛应用于计算机视觉等其他领域[6]。大模型展现出的涌现能力与泛化性能,将模型训练规模推升至新量级。

基于规模定律(scaling law)的研究发现,随着模型参数规模的指数级扩展,其在各类任务上的性能也往往呈现出相应的提升趋势[7]。当前,大规模语言模型正处于参数急速扩张的时代,其研发与训练成本也随着参数数量的激增而显著上升[8]。例如,GPT-3[4]、BERT[9]、Grok-1① 等模型通常包含数十亿甚至上千亿个参数,谷歌 PaLM-2[10]更是以 3.6 万亿的参数量刷新记录。

这些模型的训练不仅要求海量数据与存储空间,还需要极高的算力支持。在训练过程中,往往需要动用数十到数百 TeraFLOPs 级别的持续运算能力[11-12]。以 OpenAI 的 GPT-4[13]为例,其采用混合专家架构(MoE)[14],在实现高效参数利用的同时也

① https://x.ai/blog/grok-os。

大幅提升了训练过程中对计算资源的需求;知名的开源大模型 Llama[15],在 2048 个 A100 80GB GPU 上,开发和训练了约 5 个月。这种参数规模的不断膨胀和算力需求的急剧增加,直接推动了训练基础设施范式的转变——传统的单卡训练模式已难以满足需求,多卡分布式训练技术正逐渐成为核心基础设施。

早期分布式训练技术大多专注于单一并行策略的优化,每种策略都只在特定应用场景下发挥出显著优势。例如,数据并行通过批量样本划分,实现简单高效的线性加速,但当模型参数达到千亿规模时,其梯度同步的通信开销激增,成为制约效率的瓶颈;模型并行可以拆分庞大的网络结构,但设备间频繁的参数通信往往导致计算效率降低、资源利用率不高;流水线并行虽然能够提高设备整体利用率,但流水线固有的"气泡"损耗又限制了训练效率的进一步提升。这种单一策略各自为政的局面,虽然满足了早期分布式训练的基本需求,但在模型规模持续扩大、训练复杂性不断增加的背景下逐渐显露出明显不足,迫使研究者开始探索多种并行策略的融合使用。

尽管 PyTorch[16]、TensorFlow[17]、JAX①等主流深度学习框架已经原生支持了一些基础并行策略,但单一策略的局限性使其无法高效应对大规模模型训练所带来的复杂挑战。为应对这一问题,近年来涌现出了一批专注于混合并行的分布式训练框架,其中最具代表性的是 NVIDIA 的 Megatron-LM②和微软的 DeepSpeed[18]。Megatron-LM 专为大规模 Transformer 模型设计,通过有效结合数据并行和模型并行策略,实现了对数百亿乃至上千亿参数规模模型的高效训练,并具备良好的集群扩展性能;DeepSpeed 则是一套综合性的分布式训练优化库,集成了混合精度训练、张量并行以及模型压缩等多种优化技术,尤其是其提出的 ZeRO 冗余优化器(Zero Redundancy Optimizer,ZeRO)[19],显著降低了多 GPU 环境中的通信与存储成本。这些混合并行框架的兴起与快速发展,正在有效解决单一并行策略难以克服的瓶颈,成为当前大模型训练的主流技术路径。

在当前超大规模模型训练不断升级的背景下,国产大模型 DeepSeek 的突破为混合并行技术提供了重要的实践样本。DeepSeek-V3[20]采用混合并行架构,融合 ZeRO-1 数据并行、流水线并行与专家并行(MoE)三种策略,在由 2048 块英伟达 H800 GPU 组成的集群上完成了对一款参数规模达到 6710 亿的混合专家语言模型的训练,全流程共消耗约 2.788M H800 GPU 小时,历时约 54 天完成预训练。相比之下,Meta 的 Llama 3.1 405B 模型在包含 16 384 块 H100 GPU 的集群上也耗时约 54 天完成预训练[21],但其算力规模高达数万 GPU 卡,凸显了 DeepSeek-V3 在算力资源利用和训练成本上的显著优势这一案例充分验证了混合并行技术在超大规模训练场景下的工程可行性,同时也展示了国内在大模型训练优化上的新进展。

由 DeepSeek 的实践案例可以看出,一个经过精心设计的混合并行策略不仅能够显著降低训练成本,还能大幅提升训练速度,但构建这样一套高效的策略并非易事。大模型的并行训练面临诸多挑战:首先,由于不同模型结构与硬件平台存在差异,每个分布式节点的训练策略(如分配每台设备训练的层数或对算子进行并行化拆分)往往需要手动调优;其次,分布式节点间的通信效率对整体训练性能影响巨大,如何实现高效的数据传输依旧是关键难题;此外,在流水线并行中,合理排布流水线以减少"气泡效应",以及在异构计算环境下充分利用不同厂商硬件特性进行任务分配,都进一步增加了大规模模型训练的复杂性。综合来看,这些因素不仅使大模型的并行训练充满挑战,也对系统协同优化提出了更高要求。

当前关于大规模深度学习模型分布式训练技术的综述研究呈现碎片化特征,既有工作或聚焦于单一并行范式(如数据并行或流水线并行)的技术演进分析[22],或局限于自动并行化等特定子领域[23],或从上层视角软硬结合进行总体分析[24-26]。这些研究虽在各自关注维度取得显著进展,但存在两个根本性局限:(1)对混合并行策略的协同机制缺乏系统性解构,导致策略组合的理论边界模糊;(2)混合并行方案的设计多依赖经验性试错,缺乏普适性的理论建模工具。特别是在 Transformer 架构主导的大模型训练场景中,现有方法在应对多维并行策略耦合时的通信-计算均衡问题仍存在显著的理论与实践鸿沟。

为系统化解决上述局限性,本文构建了一个三维分析体系:从基础策略的解构,到混合方案的优化,再到自动搜索机制在混合并行中的应用。区别于传统分类方法,本综述的创新点主要包括:

① http://github.com/jax-ml/jax。
② https://github.com/NVIDIA/Megatron-LM。

(1)建立了基于拆分的并行策略抽象框架,用于全面梳理现有的并行策略,明确其适用场景及优势。

(2)梳理了主流混合并行技术的演进历程,提出了统一且灵活的混合并行数学表达,并拓展了自动并行搜索的解空间。

(3)指出了大模型混合并行中主要的三个挑战,包括通信-计算均衡、异构计算协同与自动并行搜索,为未来研究提供了新的方向。

2 研究现状与挑战

当前大模型并行训练的研究已相当丰富,主流的并行策略如图 1 所示,包括数据并行、模型并行、张量并行、流水线并行、专家并行和序列并行。

早期的深度学习主要依赖数据并行来提高训练效率。然而,随着模型规模的急剧增长,单个计算设备的内存已难以承载完整的模型参数,旨在切分模型本身的模型并行技术应运而生。模型并行主要以两种形式存在:张量并行与流水线并行。张量并行专注于模型内部的单个算子,对大尺度矩阵运算等操作进行切分;而流水线并行则着眼于模型的宏观结构,将整个计算图按层或模块拆分为多个阶段。近年来,新的模型架构和应用需求进一步丰富了并行策略:混合专家模型的流行催生了专家并行,而大语言模型对长序列处理能力日益增长的需求,也推动了序列并行技术的诞生与应用。

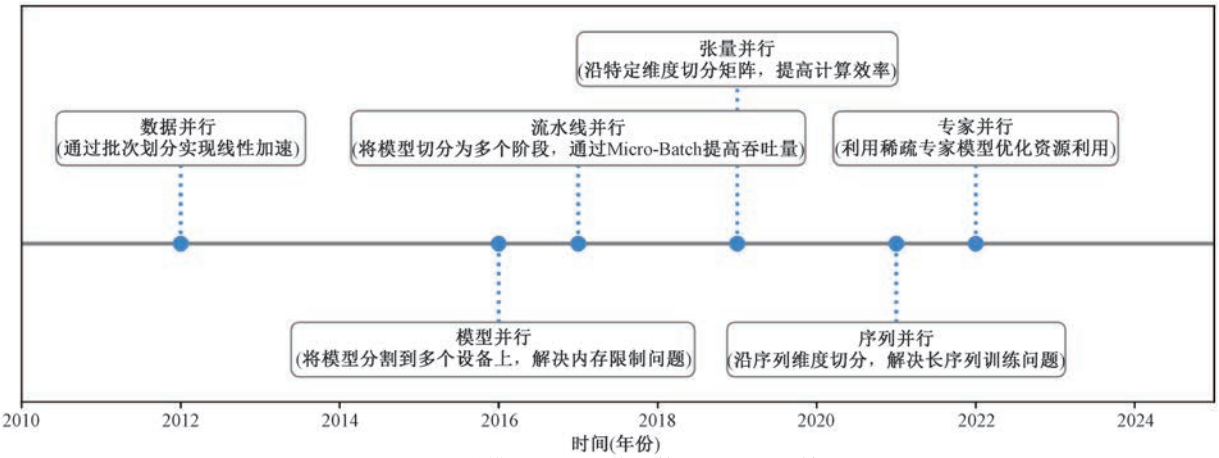

图 1 大模型训练的并行策略发展时间轴

需要注意的是,在并行训练过程中,提升训练速度与保持训练效率之间往往存在一定的权衡。训练速度指的是完成模型训练所需的时间,而训练效率则侧重于衡量计算资源的利用率和训练成本。高效的训练不仅要求缩短训练时间,还应最大化计算资源的利用率,尽量减少不必要的开销。因此,在实际应用中,应根据模型结构、硬件资源和训练需求,综合考虑不同并行策略带来的速度和效率差异,选择最合适的方案。

以下是各并行策略的简要介绍:

(1)数据并行(Data Parallelism[27], DP):将一个完整的数据批次拆分为多个子批次,分别在不同的计算单元上进行独立计算以实现并行。

(2)模型并行(Model Parallelism[28], MP):将模型自身的结构(例如层、参数矩阵)切分并部署到多个计算单元上以实现并行。它是一类并行策略的总称,其主要实现包括张量并行与流水线并行。

(3)张量并行(Tensor Parallelism[29], TP):模型并行的实现方式之一。在进行矩阵运算时,将矩阵(即张量)沿特定维度进行切分,使得多个计算单元可以并行完成同一层内不同部分的计算,通常需要额外的通信来同步结果。

(4)流水线并行(Pipeline Parallelism[30], PP):模型并行的实现方式之一。将模型的计算图纵向(通常是按层)划分为若干连续的阶段(stages),不同阶段在不同计算单元上顺序执行。通过将训练数据分批次(micro-batches)送入流水线,使得多个阶段可以并行处理不同批次的数据,从而提高设备利用率。

(5)序列并行(Sequence Parallelism[31], SP):针对处理长序列数据的场景,沿输入数据的序列维度进行拆分,各计算单元负责计算部分序列的表示,以此实现并行,减少单设备上的内存占用和计算量。

(6)专家并行(Expert Parallelism[32], EP):主要针对混合专家(MoE)模型,将不同的专家网络分配到不同的计算设备上。在计算过程中,输入数据会根据路由机制被导向相应的专家,并通过 All-to-All 等通信方式聚合结果。

随着并行策略的不断演进,可用于并行训练的方案日益增多,通常需要将多种策略组合成混合并行[33]。传统的混合并行方案往往依赖专家依据模型结构与硬件特性进行精细的手动优化,这不仅耗时耗力,也难以适应不断变化的模型和硬件环境。随着模型规模和集群规模的不断扩大,手工设计高效的混合并行策略变得愈发棘手。

为应对这一挑战,自动并行[34]技术应运而生,其目标是基于模型和硬件资源的特性自动生成最优或近似最优的并行策略。然而,由于并行策略众多且相互关联,加之设备集群规模的持续扩大,使得自动并行策略的搜索问题变得异常复杂,往往属于NP难问题。尤其在多种并行策略混合使用的情况下,策略搜索空间呈指数级增长,进一步增加了优化难度。此外,硬件异构性也使得为不同设备组合寻找最优策略更加具挑战性。因此,自动并行系统需要结合传统算法、启发式算法以及基于机器学习的优化方法,逐步逼近最优解,确保在多样化的计算资源和网络结构下均能搜索出高效的并行训练方案。

以下是大规模并行训练中面临的一些具体挑战:

(1)通信开销:在多节点并行训练中,各节点之间需要频繁交换参数和梯度信息,这会产生额外的通信开销。如果通信延迟较高,会导致整体训练速度的下降,从而影响训练效率。有效的通信策略和优化算法对于减少通信开销至关重要。

(2)负载均衡:不同计算节点的计算能力和负载情况可能存在差异,如何合理分配计算任务以确保各节点工作负载均衡是一个重要问题。负载不均衡会导致某些节点成为瓶颈,降低整体计算资源的利用率,从而影响训练效率。

(3)内存管理:随着模型参数和数据量的增加,内存管理变得越来越重要。如何高效地利用内存资源,避免内存溢出和过度的内存交换,也是并行训练中需要解决的问题。内存利用率的优化直接影响训练效率。

(4)自动并行:如何自动化设计高效的并行训练算法,充分利用多节点和多GPU资源,提高训练速度和训练效率,是并行训练的核心问题。有效的并行训练算法可以减少通信开销、提高负载均衡、增强容错性,从而实现高效的模型训练。

(5)容错:在大规模分布式训练中,节点故障是难以避免的。如何在节点故障发生时快速恢复训练过程,确保训练任务的连续性和结果的可靠性,是并行训练面临的另一个重大挑战。需要设计有效的容错机制以保证系统的鲁棒性,从而提高训练效率。

(6)异构训练:如何有效利用异构计算资源(如CPU、GPU、TPU等)进行并行训练,充分发挥各种计算设备的优势,提高训练速度和训练效率,是并行训练的又一个挑战。异构计算资源的合理配置和优化对于提高训练效率至关重要。

目前,主流的大型语言模型[4,15,35-36]几乎都采用Transformer架构。因此,本文主要聚焦于Transformer模型的并行训练方法,通过研究和优化其训练策略,为大规模模型的训练提供有益的参考和借鉴。同时,其他类型的模型在并行训练时也可以借鉴本文总结的方法和思路,以实现更高效的训练效果。

3　并行策略的抽象

现代深度学习模型的训练过程可形式化描述为对计算图 $G=(V,E)$ 的迭代优化,其中顶点集 $V=\{v_i\}_{i=1}^{n}$ 表示计算算子,边集 E 表示算子间的数据依赖关系(如残差连接),如图2(a)。无论模型架构差异如何,其本质均在执行如下循环过程[37]:

$$\theta_{t+1}=\theta_t-\eta\nabla_\theta\mathcal{L}(G(x;\theta_t))$$

其中,η 为学习率,而 $\nabla_\theta\mathcal{L}$ 则是利用自动微分构建的反向传播计算图。在这一过程中,正向计算图驱动模型预测,而反向计算图用于梯度传播,为整个训练流程中的并行策略抽象提供了理论基础。以Transformer架构的GPT3模型为例,其计算图包含96个串行Transformer层,每个层内的计算可分解为

$$\mathrm{FFN}(\boldsymbol{x})=\boldsymbol{W}_2\sigma(\boldsymbol{W}_1\boldsymbol{x}+\boldsymbol{b}_1)+\boldsymbol{b}_2,$$

$$\text{Self-Attention}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})=\mathrm{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V}。$$

在挖掘计算图 G 的并行性时,并行策略可从两个维度展开:在算子内部进行分解,或是在算子之间进行调度。基于此,Alpa[38]首次系统性地提出了算子内并行(intra-op)与算子间并行(inter-op)的二分法。算子内并行针对单个算子内部进行数据或计算的拆分,通过多个设备协同计算同一个算子,以提升算子自身的执行速度,常见如张量并行与序列并行均属此类;而算子间并行则是将整个计算图进行拆分,拆分后不同的算子或算子组(如一个网络层)作为调度单元,分配至不同设备并行执行,其典型代表是流水线并行。这两种模式达成并行的本质区别在于:算子内并行通过多设备同时执行同一个算子或算子组实现,算子间并行通过多设备同时执行不同

的算子或算子组实现。

算子内并行与算子间并行二分法的核心价值在于其抽象与简化能力。它将复杂多样的并行技术统一归纳为两个正交且互补的基础维度。这种分类降低了理解、描述和设计大规模混合并行策略的复杂性。下文将基于此理论框架，一方面深入剖析并行的本质，即如何通过不同层面的拆分诱导并行性，另一方面系统探讨两类并行之间的耦合关系，为理解和设计更高效的混合并行方案提供基础。

并行性的根源在于计算图与算子的拆分，但由于数据依赖性的存在，单纯的算子内或算子间拆分并不总能直接带来相应的并行性。例如，算子内拆分往往需要结合通信原语才能形成真正的算子内并行；而计算图拆分由于目前的模型架构呈串行特征，缺乏天然并行的算子，只有在结合算子内拆分并形成流水线的情况下，才能真正实现算子间并行。因此，本文会将拆分与并行两个概念分离，并从拆分的角度出发，分析不同拆分方法如何直接或间接地带来并行性。

首先是算子内拆分与算子内并行[19,29,39]，如图2(b)，该方法旨在挖掘算子内部的并行性。通过将每个算子拆分，来提升算子的计算速度。例如，对于Transformer中的前馈神经网络层(Feed Forward Neural Network layer，FFN)中的某一 $GEMM \in V$，输入部分涉及三个维度：[数据批次大小(B)，序列长度(S)，隐藏层维度(H)]，而模型部分涉及两个维度：[隐藏层维度1($H1$)，隐藏层维度2($H2$)]。其算子内切分可建模为：

$$T_k : \Re^{B\times S\times H} \rightarrow \prod_{i=1}^{k} \Re^{B/k_b \times S/k_s \times H/k_h},$$

其中，切分因子(k_b，k_s，k_h)分别对应数据、序列和张量维度的拆分。虽然这三维切分在理论上可能不能直接实现完全并行，但在引入必要的通信操作后，便可分别对应数据并行、序列并行和张量并行策略，从而实现整体并行效率的提升。

其次是基于计算图拆分实现的流水线并行[40-44]，如图2(c)，这一方法旨在挖掘整个计算图的全局并行性。具体做法是将完整的计算图分解为多个存在计算依赖关系的子图，同时结合对算子内部的拆分，使得一些依赖关系不再严格，从而实现流水线式的调度。以Transformer为例，通过对数据并行(DP)维度进行切分后，某些算子之间不再存在严格的前后依赖，即在一个数据批次中，每条数据在经过某个算子后，其后续计算可以独立于其他数据进行。基于这一特点，目前的流水线并行策略往往将一个完整的批次划分为多个微批次，通过在不同流水线阶段并行处理不同微批次来提高整体计算效率。

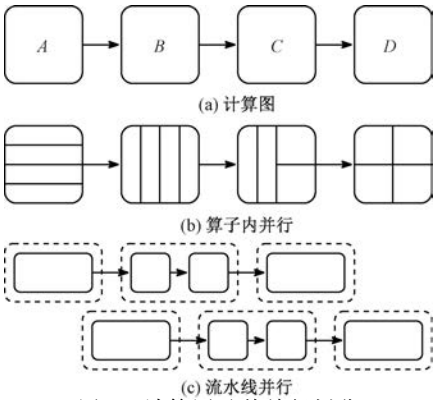

图2 计算图及其并行划分

上述并行性抽象不仅为理解各类并行策略提供了明确的理论框架，也为进一步的优化设计奠定了基础。接下来的讨论中，所有并行策略的分析都将基于这一系列抽象展开，逐步解析当前主流方法的优劣与适用场景。

3.1 算子内拆分与算子内并行

算子内拆分根据依赖关系的保持程度可分为严格依赖与非严格依赖两类。在严格依赖的算子内拆分过程中，原有计算图中的算子间依赖关系保持不变。这意味着，所有依赖于该算子的其他算子仍需等待所有子算子的计算完成，确保依赖关系的严格性。非严格依赖的算子内拆分不同于严格依赖的算子内拆分，其核心在于算子拆分后部分解除了前后算子之间的依赖关系。例如，在沿数据批次维度进行拆分时，后续算子可以在不必等待当前算子的所有子算子计算完成的情况下开始执行。

通常，算子内拆分可直接实现算子内并行。对于严格依赖型拆分，只需在必要处插入通信操作以协调数据传输，而非严格依赖型拆分则能天然实现并行执行，无需额外操作。值得注意的是，某些情况下的算子内拆分(如因果掩码的序列维度拆分)虽然不能直接带来并行性，但其非严格依赖特性可以与后续的计算图拆分相结合，从而支持流水线并行。

3.1.1 严格依赖的算子内拆分

(1)参数矩阵拆分与张量并行

张量并行是大模型时代的重要产物，其核心竞争力在于其天然将模型参数 $W \in R^{m\times n}$ 分解为 $\{W_i\}_{i=1}^{k}$ 并分布到多个设备，从而解决了单一设备存储超大模型参数的难题。其基本思想是将矩阵乘

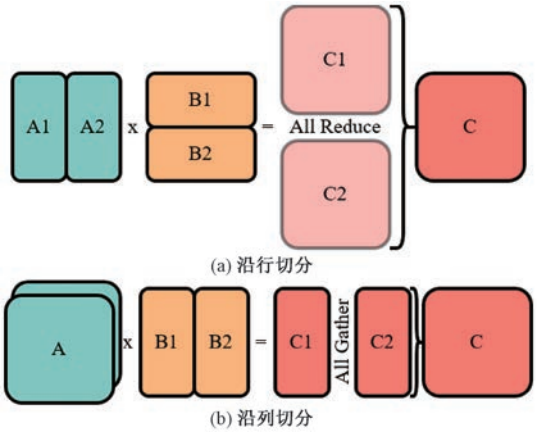

图 3　张量并行

法等计算操作拆分成多个较小的矩阵乘法:例如,可沿模型参数矩阵的行或列进行切分,在多个处理单元上并行计算,最后汇总各单元的计算结果。该过程可形式化描述为

$$\boldsymbol{W}=\overset{k}{\underset{i=1}{\oplus}}\boldsymbol{W}_i,\boldsymbol{W}_i\in R^{m/k\times n}\ \text{or}\ \boldsymbol{W}_i\in R^{m\times n/k},$$

$$\boldsymbol{Y}=\overset{k}{\underset{i=1}{\cup}}\boldsymbol{X}_i\boldsymbol{W}_i,\quad \boldsymbol{X}_i=\mathcal{T}_{\mathcal{k}}(\boldsymbol{X})。$$

其中,$\mathcal{T}_{\mathcal{k}}$ 表示输入张量切分操作,如 Scatter、Broadcast,$\cup$ 表示计算结果聚合操作,如 All-Gather、All-Reduce。

当沿行维度切分 $\boldsymbol{W}$ 时,如图 3(a),在这一过程中,$\boldsymbol{W}_i\in R^{m/k\times n}$ 预先分布式储存在不同的设备上,$\boldsymbol{X}_i\in R^{b\times s\times m/k}$ 需要通过 Scatter 操作将输入数据切片并分发到所有计算单元,每个设备计算局部结果 $\boldsymbol{Y}_i=\boldsymbol{X}_i\boldsymbol{W}_i$,计算完成后通过 All-Reduce 操作来整合结果。

$$\boldsymbol{Y}=\text{All-Reduce}\left(\sum_{i=1}^{k}\boldsymbol{X}_i\boldsymbol{W}_i\right)。$$

当沿列维度切分 W 时,如图 3(b),在这一过程中,$\boldsymbol{W}_i\in R^{m\times n/k}$ 预先分布式储存在不同的设备上,$\boldsymbol{X}_i=\boldsymbol{X}$ 需要通过 Broadcast 操作将输入数据完整地分发到所有计算单元,每个设备计算局部结果 $\boldsymbol{Y}_i=\boldsymbol{X}_i\boldsymbol{W}_i$,计算完成后通过 All-Gather 操作来整合结果。

$$\boldsymbol{Y}=\text{All-Gather}\left(\sum_{i=1}^{k}\boldsymbol{X}_i\boldsymbol{W}_i\right)。$$

朴素的切分策略引入了额外的通信开销,包括 Scatter、Broadcast、All-Gather 和 All-Reduce 等操作,这些通信过程可能成为训练效率的瓶颈。Megatron-LM[29] 通过优化 Transformer 的注意力层和前馈神经网络层,显著降低了通信开销。在多层感知器(MLP)层中,权重首先沿列切分,然后再沿行切分。这种方法天然地避免了列切分后所需的 All-Gather 操作以及行切分前的 Scatter 操作,从而减少了通信负担,提高了并行计算的效率。对于注意力层,其多头注意力机制本身具备良好的并行性,可以在多个处理单元上直接进行并行计算。在从子空间到隐藏层的映射过程中,每个头对应的输入数据已经自然切分,使得参数矩阵可以直接沿行切分,无需额外的通信操作。整体流程如图 4(a),其中 f 表示在前向时进行 Broadcast,反向时进行 All-reduce,$\bar{f}$ 则是将 f 的两次通信操作取反。

(2)输入序列维度拆分与序列并行

大型语言模型(LLM)在训练过程中常常因序列过长而导致显存溢出(OOM)问题。为了解决这一难题,序列并行(SP)技术应运而生,其核心思想是沿序列维度对输入数据进行切分。

由于注意力机制(attention)的计算依赖于整个序列信息,Korthikanti 等人[39] 最初仅将序列并行技术应用于那些在序列维度不存在计算依赖的算子(例如 Dropout 层和 LayerNorm 层),并将其作为 Megatron 张量并行的扩展。这一方法不仅进一步挖掘了更多算子的内部并行性,还将每个 Transformer 层的通信需求从原先的两次 Broadcast 和两次 All-Reduce 优化为两次 All-Gather 和两次 Reduce-Scatter,其整体流程如图 4(b)所示,其中 g 表示在前向时进行 Reduce-Scatter,反向时进行 All-Gather,$\bar{g}$ 则是将 g 的两次通信操作取反。但这种方案将序列并行的维度与张量并行的维度耦合在一起,从而限制了序列并行的扩展性。

为了解决 Attention 算子在序列维度上无法直接并行计算的问题,Hao Liu 等人提出了一种新的 Attention 算法——Block-wise Attention[45]。该算法本质上是 flash-attention[46-47] 的分布式版本,并得益于 online softmax 算法[48],将 Attention 的空间复杂度从 $O(s^2)$ 降低到 $O(s)$,从而可以在序列维度上实现并行计算,极大地提升了序列并行的效率。在此基础上发展出的 Ring-Attention 算法[49] 更进一步优化了计算与通信的重叠。其核心算法描述如下:

$$\text{Attn}(\boldsymbol{Q}_i,\boldsymbol{K}_j,\boldsymbol{V}_j)=\frac{e^{\boldsymbol{Q}_i\boldsymbol{K}_j^T-\max(\boldsymbol{Q}_i\boldsymbol{K}_j^T)}}{\sum e^{\boldsymbol{Q}_i\boldsymbol{K}_j^T-\max(\boldsymbol{Q}_i\boldsymbol{K}_j^T)}},$$

$$\max_i=\max(\max(\boldsymbol{Q}_i\boldsymbol{K}_1^T),\cdots,\max(\boldsymbol{Q}_i\boldsymbol{K}_B^T)),$$

$$\text{Attn}(\boldsymbol{Q}_i,\boldsymbol{K},\boldsymbol{V})=\left[e^{Q_iK_j^T}-\max_i\text{Attn}(\boldsymbol{Q}_i,\boldsymbol{K}_j,\boldsymbol{V}_j)\right]_{j=1}^{B_{kv}}。$$

其中,$\boldsymbol{Q}_i$ 表示第 i 个 Query 向量,$\boldsymbol{K}_j$ 与 $\boldsymbol{V}_j$ 分别为

第 j 个 Key 和 Value 向量，B_{kv} 为 Key 与 Value 的分块数。

基于 ring-attention 技术，Megatron-LM 项目进一步开发了上下文并行(Context Parallel，CP)。作为对 Megatron SP 的扩展，不同于其 SP 仅适用于 Dropout 层和 LayerNorm 层的方案，CP 同时也可扩展至 Attention 层和前馈神经网络层(FFN)，从而成为与张量并行(TP)正交的一种并行维度。其整体流程如图 4(c)所示，其中 h 表示在前向时不用进行通信操作，反向时需要进行 All-Gather。

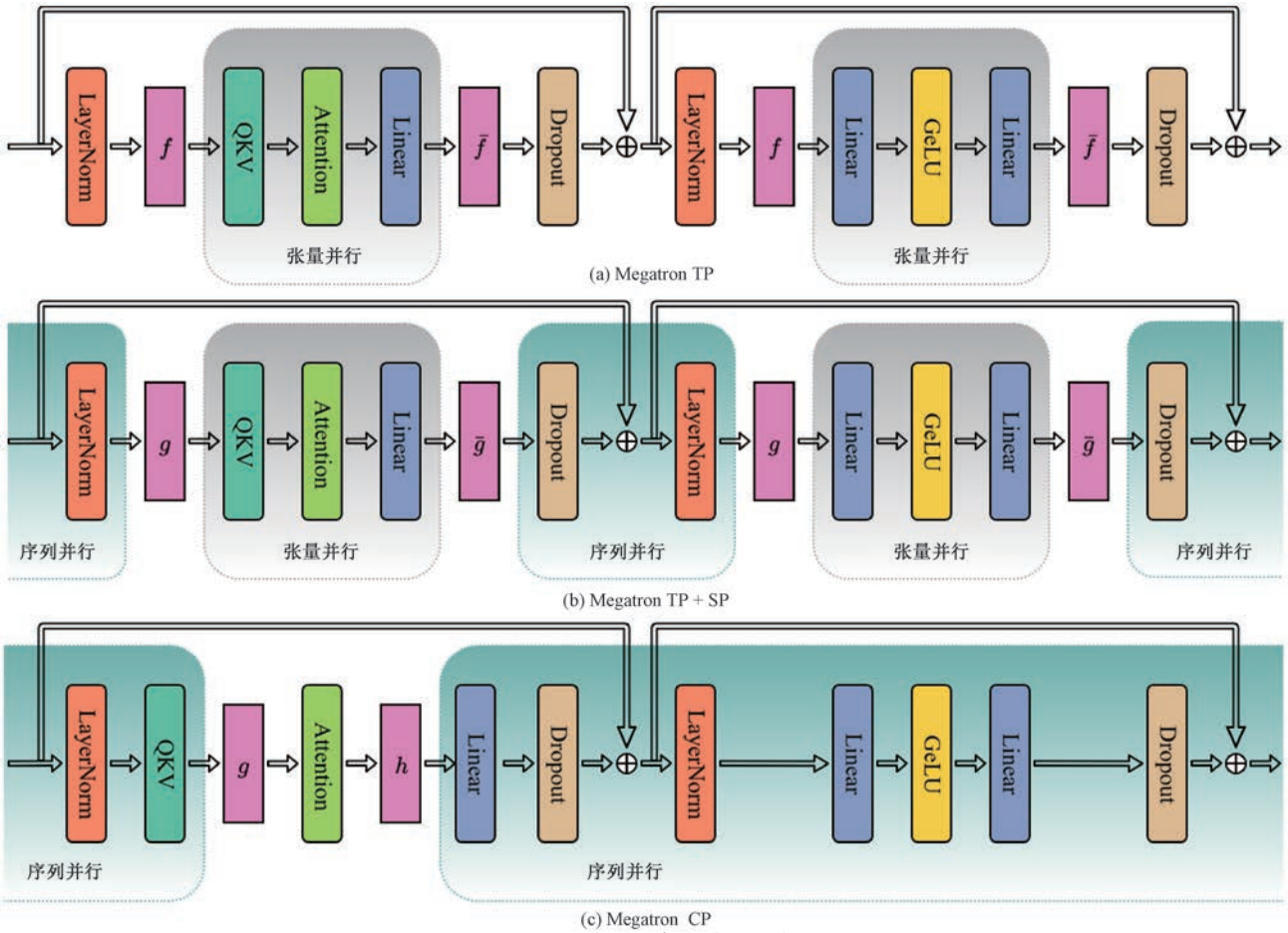

图 4 张量并行与序列并行示意图

与此同时，Deepspeed 团队在序列并行策略上也提出了自己的解决方案——DeepSpeed-Ulysses[50]。该方法核心在于沿序列维度对输入数据进行划分，然后利用高效的 all-to-all 集合通信来交换 QKV 矩阵。在进行 attention 计算前，每个 GPU 通过 all-to-all 通信交换数据，以获得完整序列中不同 attention head 的子集；计算完成后，再次利用 all-to-all 通信将结果按序列维度重新划分。与 ring-attention 相比，这种设计显著降低了通信开销。理论分析表明，当通信量与序列长度及 GPU 数量成正比时，该方法能保持通信开销恒定，从而有望扩展到百万级 token 的序列长度。实验结果也表明，该方法能以超过现有最佳方案 2.5 倍的速度训练 4 倍更长序列长度的模型，并且具有良好的通用性和易集成性。不过，该方案也存在一定局限性，即序列并行维度的大小必须能被 head 数量整除，在 GQA(Grouped-Query Attention)[51] 方案下，head 数量的限制问题更为严苛。

实际上，ring-attention 与 Ulysses 两种方案本质上是正交的，已有部分工作尝试将二者结合[52]，以进一步提升模型训练效率。

(3)专家拆分与专家并行

专家并行[53-57](Expert Parallelism，EP)是一种专为稀疏专家模型[14,58](Mixture-of-Experts，MoE)设计的并行策略，其核心思想在于将模型中的各个专家模块分布到多个计算设备上，从而实现资源的高效利用。在这种架构下，每个专家仅负责处理输入数据的一个子集，即在整个训练过程中只有部分专家被激活，从而显著降低了单个设备的计算负担。其核心原理可概括为

$$\mathcal{M}(\boldsymbol{x})=\sum_{i=1}^{E}G_i(\boldsymbol{x})E_i(\boldsymbol{x}),E_i\in\{\text{Device}_k\}_{k=1}^{K}。$$

其中,$G_i(\boldsymbol{x})$ 为门控网络生成的稀疏激活权重,E_i 为分布式部署的专家模块。该架构通过 All-to-All 通信实现跨设备数据交换,在 4096 卡集群中可支持万亿级参数训练[59]。然而,该技术仍面临以下关键挑战:

①通信开销与扩展性瓶颈

All-to-All 通信机制虽能保证数学等价性,但其通信复杂度随设备规模呈超线性增长。以设备集群数量 K 为例,传统 All-to-All 的通信时延为

$$T_{\text{comm}}=\alpha\log K+\frac{2SB}{K\beta}。$$

其中,α 为网络延迟,β 为带宽,S 为序列长度,B 为批尺寸。当 $K=4096$ 时,通信开销可占总训练时间的 38%以上。近期研究提出分层 All-to-All 策略,通过节点内 NVLink 聚合与节点间 InfiniBand 传输的协同优化。如 DeepSeek 开源的 DeepEP[60],与 DeepSeek-V3[20]论文中提出的组限制门控算法保持一致,DeepEP 提供了一组针对非对称域带宽转发进行了优化的内核,例如将数据从 NVLink 域转发到 RDMA 域。这些内核提供高吞吐量,使其适用于训练和推理预填充任务。

②动态负载均衡困境

尽管 EP 通过门控网络实现专家选择的动态调整,但局部负载均衡(Micro-batch Level)会导致专家特异化受限。阿里云团队在训练 Qwen 系列模型时研究发现[61],当单个微批次数据来源单一时(如全为代码数据),传统负载均衡损失函数会强制均匀分配输入,抑制专家领域特化能力。其提出的全局均衡方法(Global Load Balance)通过在 128 个连续微批次上统计专家激活频率,使领域特异性专家比例提升 47%,同时在 BigBench 基准上降低困惑度(PPL)达 15%。

3.1.2　非严格依赖的算子内拆分

(1)数据批次维度拆分与数据并行

数据并行(Data Parallelism,DP)[62]的核心思想是将整个数据集拆分成多个批次,并在不同的计算单元上运行相同的模型副本。每个计算单元负责处理其中一部分数据,独立完成前向传播和反向传播操作。反向传播结束后,各计算单元会通过 All-Reduce 操作汇总梯度,以确保模型参数的一致性。数据并行因其实现简单且能充分利用多设备的计算资源,特别适合处理大规模数据集。但在大规模集群中,All-Reduce 操作可能成为显著的性能瓶颈。

传统的数据并行方式存在存储冗余问题,如模型参数和优化器状态会被重复存储多份,而 ZeRO-DP[19]则在此基础上进行了改进。ZeRO 提供了三个递进的优化阶段:ZeRO-1 仅对优化器状态进行分片,在使用 Adam 等高阶优化器时,这部分通常是模型参数量的数倍;ZeRO-2 在此基础上,进一步对梯度进行分片;而 ZeRO-3 则实现了最彻底的优化,将模型参数本身也进行分片。这三个阶段为不同规模的模型和硬件环境提供了从显著到极致的显存优化选择,但其显存收益伴随着不同程度的通信开销。

在通信层面,ZeRO 各阶段的开销差异显著。ZeRO-1 的优势在于,它仅在优化器状态更新时才涉及本地状态的计算,在训练的前向和反向传播过程中不引入任何额外的通信。相比之下,ZeRO-2 在反向传播时,会通过梯度分桶技术,在每个梯度桶计算完成后立即触发一次 Reduce-Scatter 操作来代替传统数据并行的 All-Reduce,该方案不会增加通信量,但通信频率和复杂度有所提升。而 ZeRO-3 则需要在每个计算层的前后都进行 All-Gather 操作来重组完整的模型参数,这带来了密集的通信需求。不同模型根据其训练框架和并行策略,会选择不同的 ZeRO 阶段,但在混合并行场景下,为了不让 ZeRO 的通信开销与原有的并行策略冲突,ZeRO-1 往往是一个稳妥的选择[15,20,35]。

(2)不构成序列并行的序列维度拆分

Token-level 拆分是一种沿序列维度进行的特殊拆分方式,适用于自注意力计算仅依赖于先前位置的 Transformer 模型,如 GPT 系列模型[3-4],这些模型采用解码器结构,使用因果掩码(causal masking)确保每个位置的输出仅依赖于当前位置及之前的位置。准确地说,对于输入隐藏状态序列 $(\boldsymbol{h}_1,\boldsymbol{h}_2,\cdots,\boldsymbol{h}_t)$,自注意力层 $\text{SelfAtt}(\boldsymbol{h}_t)$ 的计算仅取决于先前位置 $(\boldsymbol{h}_1,\boldsymbol{h}_2,\cdots,\boldsymbol{h}_{t-1})$ 的隐藏状态,并且前馈层 $\text{FFN}(\boldsymbol{h}_t)$ 的计算仅取决于 $\boldsymbol{h}_t$ 本身。不同于之前讨论的序列并行和上下文并行,由于自注意力的计算依赖于序列中的所有 Token,导致后续算子必须等待当前算子的所有子算子完成计算。然而,对于这种特殊的模型,得益于其仅依赖于先前位置的特性,沿序列维度拆分后,计算图内的各算子可以在不等待前一个算子全部完成的情况下继续执行。

3.2　计算图拆分与流水线并行

计算图拆分是指将模型划分为多个阶段,并在不同的计算设备上执行。具体来说,模型的前向传播和反向传播过程被划分为多个子图,每个子图在

不同的计算设备上并行执行，从而扩展模型的规模。计算图拆分不仅能够有效分散计算任务，还能显著减轻单个设备的内存压力。通过将模型划分为多个子图，单个设备只需存储并处理一部分计算图，从而避免了将整个模型加载到单一设备内存中的需求，特别是在处理超大规模模型时，能够有效避免内存溢出的问题。然而，单纯的算子间拆分并不能实现真正的并行性。如图 5(a)所示，如果仅将模型的不同层分配到不同的计算设备上，由于算子间的依赖性，往往会导致每个时刻只有一个设备在计算，从而造成计算资源的浪费并降低并行效率。

为了有效发掘算子间的并行性，计算图拆分后需要结合非严格依赖的算子内拆分，以实现流水化。当前主流的流水线并行通常基于数据批次维度拆分。在首次提出流水线并行的 GPipe[40] 中，通过将一个数据批次拆分为多个微批次，不同的微批次在所有计算设备上并行计算，从而能够在每一步训练过程中达到近似流水线并行的效果。GPipe 的流水线结构如图 5(b)所示。通过这种方式，计算图拆分与流水线并行相结合，可以在不显著增加内存负担的情况下，实现更高效的并行计算。需要注意的是，本文仅讨论同步的流水线并行，异步流水线(如将多步训练流水化而不进行梯度同步)由于存在数学不等价的问题，因此不在本文讨论范围内。

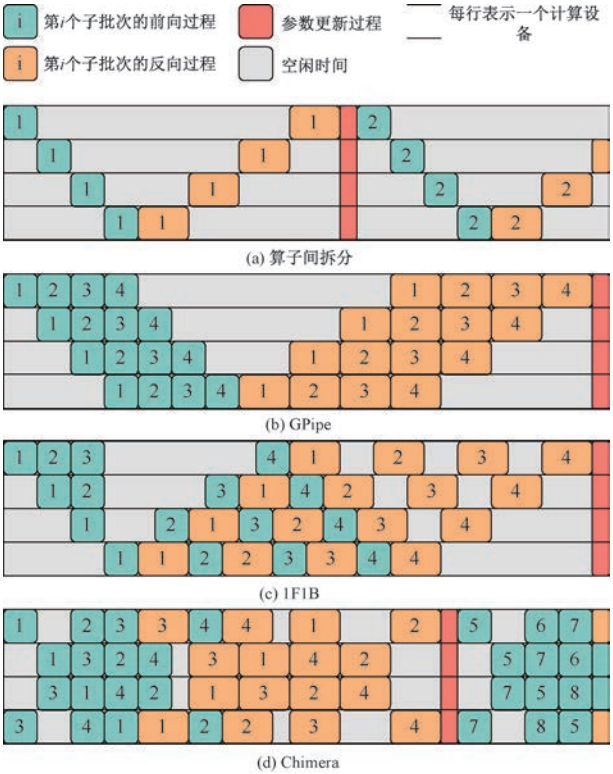

图 5 流水线并行

流水线中的各个计算阶段之间存在数据依赖性，导致某些设备在等待其他设备完成计算时处于空闲状态，无法有效利用计算资源，这些阶段性的空闲时间即为 Bubble。目前有关流水线并行的研究重点在于如何最小化 Bubble 的数量和持续时间，同时降低显存的占用。例如，通常采用异步通信方式，以减少通信开销，即每个计算设备在完成自身微批次的计算后，立即进行通信并启动下一个微批次的计算。这样能够在一定程度上重叠通信和计算过程，从而进一步减少流水线中的 Bubble，提升整体并行效率。

Narayanan 等人在 PipeDream[41] 中提出的 1F1B 思想，通过重排流水线来尽早进行反向计算，从而形成一个前向过程和一个反向过程交错进行的流水线结构。虽然这种方法并未缩短 Bubble 的持续时间，但提前进行反向过程意味着可以更早地释放前向过程产生的激活值，使得这种流水线在峰值显存占用方面表现更为优异，目前 1F1B 流水线已经被广泛应用于各种策略之中[42]。

此外，流水线并行还有许多优化工作，例如 GEMS[63]、CHIMERA[64] 提出双发射流水线，ZERO-BUBBLE[43] 创新性地提出了将反向过程中的模型参数梯度计算和激活值梯度计算分离，进一步重排流水线，减少 bubble。同时，他们还提出每个计算设备持有两块子图，让整个训练在流水线上以 V 形方式进行，解决了 1F1B 策略可能导致的显存利用率不均问题的同时几乎消除了流水线 Bubble。同时，流水线结合 offload[65]、Checkpoint 等技术也能解决流水线并行的显存问题。

除了沿着数据批次维度拆分，还可以沿着 Token 维度拆分。TeraPipe[44] 很好地利用了 Transformer 的特性，即较长的序列需要较长的计算时间。不是以微批为单位将数据馈送到管道，而是沿着令牌轴(即序列轴)不均匀地分割序列数据，然后将它们馈送到管道中，其中每次分割具有相似的执行时间。

3.3 并行计算与通信

不论是通过拆分算子或是拆分计算图，将训练任务并行化后，均不可避免会引入额外的通信。通信开销通常是并行训练的主要瓶颈之一，尤其在大规模分布式训练中，通信延迟和带宽的限制可能会显著影响整体训练效率。Flux[66] 中提到，在使用 PCIe 设备进行训练时，张量并行引入的通信时长占整个训练过程的 40%～60%。因此，在设计并行计

算架构时,需要充分考虑通信开销,并采取相应的优化策略来降低通信成本。

在进行算子内并行时,往往需要引入额外的集合通信操作,以确保各个计算单元之间的数据一致性和正确性。这些通信操作包括但不限于 Scatter、Broadcast、All-Gather 和 All-Reduce 等。每种通信操作都有其特定的开销和适用场景。例如,Scatter 操作用于将数据从一个设备分发到多个设备,而 All-Gather 则用于将各个设备上的数据聚合到一起。在流水线并行中,通信开销则是阶段之间的 P2P 通信。每个阶段的计算结果需要通过通信操作传递给下一个阶段。

在优化通信开销时,可以从两方面入手:(1)优化通信算子本身的速度与效率;(2)重叠计算与通信,掩盖通信开销。前者主要是通过改进集合通信算法、数据压缩与编码等方式来减少通信延迟和带宽占用;后者则是通过异步通信、算子融合等方式来减少通信对计算的阻塞影响。具体如下:

(1)集合通信算法优化:通过改进集合通信算法(如使用更高效的 All-Reduce 算法)来减少通信延迟和带宽占用。为降低通信延迟和带宽占用,研究者提出了多种优化的 AllReduce 算法。经典的 Ring AllReduce 通过将节点组织成环形拓扑,采用 Reduce-Scatter 和 AllGather 两个阶段,实现了带宽最优的通信模式,已被广泛应用于 Horovod 等框架中[67]。Hierarchical AllReduce 则将节点划分为多个组,先在组内进行 AllReduce,再在组间进行通信,最后将结果广播回各组,显著减少了通信步骤,提升了大规模集群的训练效率[68]。此外,2D-Torus AllReduce 利用二维网格拓扑,在两个维度上并行执行通信操作,进一步降低了通信开销,适用于超大规模分布式系统[69]。近年来,针对特定挑战的优化算法也不断涌现。例如,StragglAR 通过在存在慢节点(straggler)时,先对其余节点执行 Reduce-Scatter,再在慢节点同步后完成 AllReduce,显著提升了训练速度[70]。GenTree[71] 则通过引入新的时间成本模型,设计出适用于树状拓扑的 AllReduce 计划生成算法,在实际测试中实现了 1.22×至 1.65×的加速。此外,利用深度强化学习自动生成适应不同网络拓扑的 AllReduce 调度策略,也成为提升通信效率的新方向[72]。

(2)数据压缩与编码:在大规模分布式训练中,通信带宽常成为性能瓶颈。为缓解此问题,研究者提出了多种通信压缩技术,包括梯度量化(如 QSGD[73] 和 signSGD[74])、梯度稀疏化(如 Deep Gradient Compression[75])、误差补偿机制(如 MEM-SGD[76])以及低秩近似(如 PowerSGD[77])。近年来,新的方法如 ACP-SGD[78]、MiCRO[79]、TAGC[80] 和 QSDP[81] 等进一步提升了训练效率和可扩展性。这些技术的结合显著推动了大规模分布式深度学习的效率和可行性。

(3)算子融合:将计算与通信算子进行融合,微软提出的 CocoNet[82],最早提出异步张量并行(Async Tensor Parallelism)的工作,作者提供了一种通用 Kernel 融合的范式,能够自动生成集合通信与常见计算操作(如 GEMM 和卷积)之间的融合 Kernel。Flux[66] 在常规算子融合的基础上更进一步,基于 Warps 异步执行,通过将内存写回操作替换为通信操作,在不降低算子效率的前提下将绝大部分所需的通信融入并隐藏在计算中,从而实现了更高效的异步通信。

(4)流水线通信重叠:通过设计流水线调度策略(如 1F1B),可以显式地重叠计算与通信。在前向计算过程中,系统可以异步发送输出激活值;在反向计算阶段,则可以异步接收上游梯度,从而将通信隐藏在计算过程中,大幅减少流水线空泡。例如,PipeDream 展示了通过异步转发激活值和梯度实现计算与通信的重叠,并且比数据并行减少高达 95% 的通信开销[41,83]。当结合 Offload 策略时,通过合理的流水线阶段安排,可以在计算前提前预取激活,实现通信-计算的掩盖。以 SSD 为目标的激活 offload 系统 SSDTrain(又称 TBA)就是一个典型示例,它通过 I/O 与 GPU 计算完全并行实现激活预取,从而减轻显存压力且不损失训练性能[84]。另有 FlashFlex[85] 等系统将激活异步 offload 至 CPU 或 SSD,并通过与计算同步的传输机制有效覆盖通信延迟。因此,结合异步通信和激活预取/加载技术的流水线调度,能够将通信隐藏于计算之下,以此提升大模型训练的整体效率。

3.4　其他相关技术

(1)Offload 技术[86-87]:大模型训练中的 Offload 技术是一种通过将部分计算数据(如模型参数、梯度、优化器状态)从 GPU 显存临时转移到 CPU 内存或硬盘,以缓解 GPU 显存不足的优化方法。其核心原理是在训练过程中动态管理存储资源:当 GPU 显存满载时,将暂时不用的数据卸载到内存或磁盘,待需要时再加载回显存,从而突破单卡显存限制训练超大模型。这种方法常与流水线并行

结合使用，通过将模型分阶段分配到不同设备的同时，利用 Offload 技术进一步降低各阶段的显存压力，并通过异步预取数据、重叠通信（如 CPU-GPU 数据传输）与计算（如 GPU 的前向/反向传播）来减少空闲等待时间，从而优化整体效率。

（2）重计算技术[88-89]：又称 Checkpoint 技术，是一种通过以计算换显存的大模型训练优化方法，核心思想是在反向传播时动态重新计算前向传播中的中间激活值，而非全程存储。具体来说，在前向过程中，仅选择性保留部分关键层的激活结果（即设置检查点），其余中间结果在使用后立即释放显存；当反向传播需要这些数据时，再根据检查点临时重算对应的中间结果。例如，将神经网络划分为若干段，每段仅保留输入输出，反向时逐段重算内部激活。这种技术可显著降低显存占用，但在全量重计算的情况下会引入约 30％的额外计算开销，适用于显存紧张但算力相对充足的场景，常与 Offload、ZeRO 等并行策略结合，在训练超大规模模型时实现显存与计算效率的平衡。

（3）异步训练[62,90]：在分布式集群中，各工作节点独立计算本地梯度后，无需等待其他节点同步，而是以异步方式将梯度推送至参数服务器，从而更新全局模型参数。在这种模式下，不同节点可能基于不同版本的模型参数计算梯度（即“过时梯度”），这会导致全局模型更新时存在版本不一致性。虽然异步更新能显著减少同步等待时间、提升硬件利用率并加快训练速度，但梯度冲突和参数过时所引入的噪声可能会影响模型收敛的稳定性。为平衡训练效率与模型一致性，实践中常通过限制版本延迟阈值、动态调整学习率或引入延迟补偿机制（如 Stale Synchronous Parallelism，SSP[91]）来加以调控。该技术尤其适用于数据吞吐量大、通信成本高的场景，但在实际应用中需在训练速度和模型一致性之间做出谨慎权衡。有研究表明，大模型对延迟更加敏感[92]。

4 混合并行技术的演进脉络

混合并行技术的演进始终围绕三个核心命题展开：

（1）大模型的可训练性：主要聚焦于如何突破显存限制，使得超大规模模型的训练成为可能。

（2）系统效率的提升：在实现模型可训练性的基础上，进一步探讨如何在资源受限的环境下优化资源利用率，以加快训练过程。

（3）自动化探索：研究如何借助程序化和自动化手段，系统性地搜索并构建更优的混合并行执行方案。

早期研究通过针对特定的模型结构组合特定的并行策略，使大型模型能够在分布式系统中进行训练。随着模型规模的扩大和模型变种的增多，出现了更强大的混合并行系统，随着系统持续更新，会对数据并行、张量并行、序列并行、专家并行和流水线并行等主流并行策略进行部分甚至全部的支持[18,93-95]。

随着模型的复杂性增加和计算设备的多样化，如何提高大型模型训练系统的效率成为一个重要课题。尽管像 Megatron-LM 这样的分布式混合并行系统功能强大，支持所有主流并行策略，但它们主要针对同构集群设计，未对异构集群或层间异构模型进行优化。为了解决这些问题，新型混合并行系统提出了新的混合并行思路，例如允许各个流水线阶段使用不同的算子内并行策略，以适应异构设备[93,96]。

在传统的混合并行策略设计中，通常依赖领域专家的经验。然而，随着模型规模的扩大和复杂性的增加，手动设计并行策略已难以应对指数级增长的解空间。为此，自动并行技术应运而生，利用自动化工具和算法，探索更优的混合并行策略，减少对人工经验的依赖，提高并行策略的搜索效率。

4.1 部分策略组合的混合并行

在混合并行发展早期，学术界和工业界提出了多种仅结合部分策略的方案：DeepSpeed-MoE[59]将数据并行与专家并行相结合，通过在每个专家子网络内部并行化路由和梯度更新，实现了数百亿参数 MoE 模型的高效训练。MixPipe[97]在同步数据并行训练中引入双向流水线并行，通过灵活注入微批次到正反两个流水线来提升设备利用率和吞吐。也有一些工作结合数据并行与模型并行[98]。早期的 Mesh-TensorFlow[99]框架则通过在多维计算网格上对张量维度进行切分，将数据并行与模型并行统一到同一编程接口中，为超大规模 Transformer 的预训练奠定了基础。

4.2 强一致性的同构流水线

经过多年的技术迭代与演进，以 Megatron-LM 和 DeepSpeed[100]为代表的现代分布式训练框架已日臻成熟。这些系统通过持续的架构升级，目前已能够全面支持包括数据并行、张量并行、序列并行、专家并行、流水线并行在内的所有主流并行策略。但这些系统通常要求所有并行策略在设备全局

正交生效,这意味每种策略都会作用于所有训练设备之上,无法实现例如前一个流水线阶段使用张量并行而后一阶段使用数据并行这样的异构配置。

为了精确描述这种并行结构,定义所有算子内并行维度的集合为 $P_{\text{intra}}=\{DP,TP,EP,SP,\cdots\}$,其中 DP,TP,EP,SP 分别代表数据、张量、专家和序列并行维度。该集合是可扩展的,允许未来包含其他算子内并行策略。混合并行策略实际应用的算子内并行策略 $\mathcal{P}_{\text{intra}}\subseteq P_{\text{intra}}$。对于任意一个算子内并行维度 $\mathcal{P}_{\text{intra}}\subseteq P_{\text{intra}}$,定义其并行度为 N_p,对应的设备索引集为 $D_p=\{0,1,\cdots,N_p-1\}$。

因此,算子内并行策略共同构成的设备拓扑子空间 $\mathcal{D}_{\text{intra}}$ 可以通过这些索引集的笛卡尔积来定义。这个拓扑子空间可以被理解为一个多维设备网格。在此网格中,每个设备都由一个唯一的坐标向量 $\boldsymbol{d}_{\text{intra}}\in\mathcal{D}_{\text{intra}}$ 标识。该向量可以表示为一个由并行维度 $\boldsymbol{d}_{\text{intra}}\in\mathcal{D}_{\text{intra}}$ 索引的族,其明确了设备在各个算子内并行维度中的角色:

$$\mathcal{D}_{\text{intra}}=\prod_{p\in\mathcal{P}_{\text{intra}}}D_p,$$

$$\boldsymbol{d}_{\text{intra}}=(d_p)_{p\in\mathcal{P}_{\text{intra}}},d_p\in D_p。$$

在此基础上,引入流水线并行维度并设其并行度为 N_{PP},对应的阶段索引集为 $D_{\text{PP}}=\{0,1,\cdots,N_{\text{PP}}-1\}$。在一个同构流水线配置中,每个流水线阶段都采用完全相同的算子内并行拓扑,即共享同一个 $\mathcal{D}_{\text{intra}}$。因此,可以得出同构流水线的全局设备拓扑空间 $\mathcal{D}_{is\sigma}$ 的数学表达:

$$\mathcal{D}_{is\sigma}=D_{\text{PP}}\times\mathcal{D}_{\text{intra}},$$

$$\boldsymbol{d}=(d_{\text{PP}},\boldsymbol{d}_{\text{intra}})。$$

$\mathcal{D}_{is\sigma}$ 是算子内拓扑子空间 $\mathcal{D}_{\text{intra}}$ 与流水线阶段索引集 D_{PP} 的笛卡尔积。通过这种正交分解,系统中的任何一个设备都可以通过一个全局唯一的复合坐标 $(d_{\text{PP}},d_{\text{intra}})$ 进行寻址。

4.3　阶段解耦的异构流水线

4.2 节所述的主流分布式训练框架在流水线并行的设计上遵循严格的强一致性同构范式。具体而言,该范式要求流水线中的每一个阶段必须继承完全一致的算子内并行配置,甚至要求其硬件拓扑与通信带宽也保持高度一致。这种设计虽然降低了并行策略生成与通信管理的复杂度,但其高效运行往往建立在一个强假设之上:即整个训练集群是由规格完全统一的计算设备所构成的理想同构环境。

然而,实际应用中,计算集群往往由性能各异的异构设备组成,如不同型号的 GPU、CPU,以及其他加速器。在这种环境下,传统的均匀划分策略可能导致计算资源利用率不均衡,进而影响整体训练效率。为应对这些挑战,新型混合并行系统如 FlagScale[93] 和 InternEvo[96] 提出了创新的并行策略。这些系统允许在不同的流水线阶段采用不同的算子内并行策略,以适应异构设备的特性,从而优化资源利用率并提升训练性能。

本节的数学表述将沿用前一节已定义的符号体系,如并行维度集 P_{intra} 与流水线阶段索引集 D_{PP}。

在阶段解耦的异构流水线系统中,每个流水线阶段 $k\in D_{\text{PP}}$ 都可以拥有独立的算子内并行配置。这包括该阶段所采用的并行策略集合 $\mathcal{P}_{\text{intra}}^{(k)}\subseteq P_{\text{intra}}$,以及每种并行策略的并行度 $N_p^{(k)}$。对于其内部的任一并行维度 $p\in\mathcal{P}_{\text{intra}}^{(k)}$,其并行度记为 $N_p^{(k)}$。相应的设备索引集与拓扑子空间定义如下:

$$D_p^{(k)}=\{0,1,\cdots,N_p^{(k)}-1\},$$

$$\mathcal{D}_{\text{intra}}^{(k)}=\prod_{p\in\mathcal{P}_{\text{intra}}^{(k)}}D_p^{(k)}。$$

其中,$\mathcal{D}_{\text{intra}}^{(k)}$ 表示第 k 个流水线阶段内,用于执行其算子的设备网格拓扑。

全局设备拓扑空间 $\mathcal{D}_{\text{hetero}}$ 是所有阶段拓扑子空间的不相交并集,每个设备通过一个复合坐标进行唯一标识:

$$\mathcal{D}_{\text{hetero}}=\bigsqcup_{k\in D_{\text{PP}}}(\{k\}\times\mathcal{D}_{\text{intra}}^{(k)}),$$

$$\boldsymbol{d}=(d_{\text{PP}},\boldsymbol{d}_{\text{intra}}),d_{\text{PP}}\in D_{\text{PP}},\boldsymbol{d}_{\text{intra}}\in\mathcal{D}_{\text{intra}}^{(d_{\text{PP}})}。$$

此处的坐标 d 是一个序对。其第一项 d_{PP} 指明了设备所在的流水线阶段;第二项 d_{intra} 是一个依赖于 d_{PP} 的向量,给出了设备在该阶段内部并行网格中的具体位置。

由于相邻流水线阶段 (k) 和 $(k+1)$ 的算子内并行拓扑与张量分片方式可能不同,因此在它们之间传递数据时,必须进行一次张量重映射操作。该操作可被形式化为一个变换函数 $\mathcal{T}^{(k\to k+1)}$,它将前一阶段输出的分布式张量映射到后一阶段所需的张量形态:

$$\mathcal{T}^{(k\to k+1)}:V^{(k)}\to V^{(k+1)}。$$

这里,$V^{(k)}$ 代表在拓扑 $\mathcal{D}_{\text{intra}}^{(k)}$ 上、呈分布式形态的张量空间。若具体到张量的逻辑形状(logical shape),该变换可表示为

$$\mathcal{T}^{(k\to k+1)}:R^{\sigma_k}\to R^{\sigma_{k+1}}。$$

其中,σ_k 和 σ_{k+1} 分别代表阶段 k 的输出张量与阶段 $k+1$ 的输入张量的逻辑维度,例如 (B,S,H)。

4.4 自动并行

自动并行是指通过程序化手段自动生成并应用于深度学习训练的一组并行策略。自动并行的研究贯穿了并行策略发展的全过程，从单一策略的自动化着手，例如 vPipe[101] 通过在线算法优化算子间划分策略，有效平衡 1F1B 流水线的显存不均问题；AdaPipe[102] 则利用不均匀重计算和更细粒度的分层策略，通过两级动态优化算法实现流水线的自动优化。

然而，由于并行策略的多样性以及每个策略维度存在的大量划分粒度，自动混合并行被公认为一个 NP 难问题[103-106]。当前有关混合并行的研究通常采用先验规则来缩小搜索空间，如通过剪枝、采样等方式降低复杂度。例如，Metis[107] 利用异构 GPU 设备的特性，将策略搜索分解为流水线阶段划分和算子内并行策略划分两个独立阶段，同时通过启发式方法结合数据并行与张量并行，有效减少了搜索空间复杂度。

在求解方法上，自动并行的搜索算法主要分为两大类：机器学习驱动和传统算法驱动（如动态规划、整数线性规划）。这些方法在评估并行策略时，通常基于性能分析（Profiling）或精确的代价模型，对并行策略的计算时间、显存占用、通信开销等多个维度进行综合评估，从而找到在特定硬件环境下的最优或近似最优解。目前主流的自动并行方法见表 1。

表 1 不同自动并行策略搜索方法的比较

名称	支持的策略	搜索方法	评估方法
HDP[111]	MP	LSTM RL	Profiling
GDP[108]	MP	Transformer RL	Profiling
HeterPS[124]	DP+PP	BRKGA 和 GNN+RL	基于代价模型的 Profiling
FlexFlow[109]	TP	MCMC	基于代价模型的 Profiling
Automap[110]	TP	MCTS 和交互网络	代价模型
vPipe[101]	PP	动态规划（KL）	Profiling
PipeDream[41]	MP	动态规划	基于代价模型的 Profiling
DAPPLE[42]	DP+PP	动态规划+ILP	Profiling
OptCNN[114]	MP	动态规划（图消除与再生）	代价模型
AccPar[116]	TP	动态规划	代价模型
Alpa[38]	TP+PP	ILP+动态规划	代价模型
Piper[115]	TP+PP	2 级动态规划	代价模型
TensorOpt[117]	DP+PP	动态规划	Profiling
Galvatron[119]	DP+MP+SP+SDP	决策树+动态规划	Profiling+代价模型
Galvatron-BMW[120]	DP+MP+SP+SDP	决策树+动态规划	Profiling+代价模型
UniAP[123]	DP+TP+PP+SDP	MIQP	Profiling+代价模型
AutoOP[121]	DP+MP+PP	MIP	Profiling+代价模型
Mist[122]	DP+MP	分层 MILP+双目标优化	Profiling+代价模型

注：MP：模型并行；DP：数据并行；TP：张量并行；PP：流水线并行；SDP：分片数据并行；SP：序列并行；ILP：整数线性规划；MIP：混合整数规划；MIQP：混合整数二次规划；MILP：混合整数线性规划；MCTS：蒙特卡洛树搜索；KL：Kullback-Leibler 散度。

在机器学习驱动的策略搜索领域，GDP[108]、FlexFlow[109] 和 Automap[110] 是三种具有代表性的方法。GDP 结合了图神经网络（GNN）与 Transformer 模型，通过预训练和微调的方式，能够一次性为整个计算图生成并行方案。在处理 8 层 Transformer 模型时，GDP 的速度比 HDP[111] 快了 16.7 倍，并且支持超过五万个节点的大规模图结构。FlexFlow 则采用随机马尔可夫链蒙特卡洛（MCMC）算法[112]，探索最优的划分策略，以确定神经网络中各操作的并行配置。尽管 FlexFlow 在灵活性方面表现出色，但在处理大规模模型时，策略搜索所需的时间较长。相比之下，Automap 基于 MHLO（MLIR[113] 编码的 XLA HLO），结合搜索与学习的方法，利用蒙特卡洛树搜索（MCTS）和机器学习的交互网络，通过缩减搜索空间并优化策略传播，实现了高效的并行策略搜索。

在基于传统算法的策略搜索方法中，OptCNN[114]、Piper[115] 和 AccPar[116] 是三个具有代表性的方案。OptCNN 提出了一种逐层并行的策略。该方法通过构建模型的计算图和集群的设备图，并利用动态规划算法为每一层确定最优的划分维度，从而实现自动并行化。Piper 来自 Fiddle 项目①，采

① https://www.microsoft.com/en-us/research/project/fiddle/。

用了两级动态规划的方法来搜索张量并行和管道并行策略。其外层动态规划算法会生成数百个 NP 难度的背包子问题,这些子问题用于计算给定超参数下子图的吞吐量。为了加速求解过程,Piper 引入了“bang-per-buck”启发式方法,显著降低了计算复杂度,使其能够在 2048 个设备上为 64 层的 BERT 模型在仅 2 小时内完成策略搜索。TensorOpt[117]提出了一个名为 Frontier Tracking (FT) 的算法能够同时考虑多个目标,寻找这些目标之间的帕累托最优解(Pareto-optimal),即成本边界(Cost Frontier)。该方法基于动态规划,以高效地探索和生成内存消耗与执行时间之间不同权衡下的并行策略。

近年来,研究者们在传统算法驱动的自动并行领域取得了进一步的进展,开发出了一系列更为先进和全面的系统。Galvatron[118-119]通过决策树分解搜索空间并结合动态规划进行策略优化,以自动识别最高效的混合并行策略。作为其扩展,Galvatron-BMW[120]进一步提升了性能,它将激活检查点(CKPT)也视为一个特殊的并行维度进行联合优化,并提出了一种专注于平衡内存和计算工作负载的双目标优化工作流。Ruifeng She 等人提出了一种基于混合整数规划(MIP)的算子级并行规划方法[121],通过将并行策略规划建模为调度优化问题,针对包含多分支结构(如 MoE)的深度神经网络自动生成优于专家设计方案的算子级分布策略。此外,Mist 系统[122]在大型语言模型的分布式训练中,通过引入“重叠+不平衡意识”的混合整数线性规划与双目标约束优化方法,实现了并行方式、激活检查点、内存卸载及冗余消除等技术的协同调优。另一项创新方法是 UniAP[123],它首次提出使用混合整数二次规划(MIQP)来统一优化层间与层内并行,以寻找全局最优的训练吞吐量。这些先进系统通常都包含性能分析器来构建精确的代价模型,并特别考虑了计算与通信重叠时的性能影响。

5　混合并行新范式

尽管目前已有众多混合并行方案,但各个方案都有自己的侧重点,不管是应用的技术或是对混合并行策略组合的定义,比如 Megatron-LM 将 DP 定义为 Micro batch 大小不变,在 Batch 维度进行拓展;而 InternEvo 则定义为在 Micro Batch 维度上进行切分。Megatron 将所有并行策略定义为正交且均匀的空间,而 FlagScale 则将各个流水线阶段视为一个个不同的分组,每个分组内可以应用不同的算子内并行策略。这些定义的不同导致了我们在描述混合并行策略时没有一个统一的标准,急需一个能囊括现有所有混合并行技术的新范式来描述混合并行策略。

同时,目前主流的混合并行策略还不够灵活,往往只能表达某些并行策略的组合,且各个并行维度之间只是简单的叠加相乘的关系,面向异构或者更复杂的 Transformer 模型变种的时候传统的混合并行策略表达能力欠佳。基于此,本文提出一种更灵活的混合并行策略表示方式,相关符号见表 2。

表 2　参数符号表

符号	描述
$D=(\mathcal{N},\mathcal{L})$	设备图,表示硬件拓扑结构
$\mathcal{N}$	设备节点集合
Device_i	第 i 个计算设备
$\mathcal{L}$	连接集合
$\text{Link}_{i,j}$	Device_i 和 Device_j 间的通信连接
$\mathcal{D}$	设备子集,$\mathcal{D}\subseteq\mathcal{N}$
$G=(\mathcal{O},\epsilon)$	计算图,表示模型的计算流程
$\text{sub}G$	计算子图,表示 G 的一部分
$\mathcal{O}$	算子集合
Op_k	第 k 个算子
ϵ	依赖关系集合
$\mathcal{S}$	算子子集,$\mathcal{S}\subseteq\mathcal{O}$
$\mathcal{G}_o$	算子内并行划分后的子算子集合
A_{op}	算子内并行策略
$\mathcal{G}_{pp}$	流水线阶段划分的 $subG$ 集合
A_{pipe}	流水线阶段划分策略
$\mathcal{L}_{pp}$	流水线排布策略
A	混合并行策略,$A=(A_{\text{pipe}},\mathcal{L}_{pp})$

5.1　混合并行系统的数学定义

(1)算子内并行划分 $\mathcal{G}_{op}$

对于某个算子 Op,其算子内并行划分 $\mathcal{G}_{op}$ 是根据特定的算子内并行策略 A_{op} 划分后的子算子集合:

$$\mathcal{G}_{op}=\{\text{sub}Op_1,\text{sub}Op_2,\cdots,\text{sub}Op_K\}。$$

其具有以下性质:

①设备映射:每个子算子 $\text{sub}Op_k$ 映射到一台设备 $\text{Device}_k\in\mathcal{D}$:

$$\forall k\in\{1,2,\cdots,K\},\quad \text{sub}Op_k\rightarrow\text{Device}_k。$$

②完整性:所有子算子全部计算完毕,才能认为原算子 Op 的计算完成。

(2)算子内并行策略 A_{op}

算子内并行策略 A_{op} 描述如何使用划分函数 f 对算子进行划分,划分函数的定义见算法 1,包括:

①划分维度选择:选择划分的维度及其先后顺序。

②递归划分规则:每次划分后,确定哪些子算子需要进一步划分。

③设备映射策略:划分完成后,如何将子算子映射到设备子集 $\mathcal{D}$ 上。

策略的输入可以是单个算子 Op 或计算图 G。如果输入是 G,则对 G 中的所有算子执行相同的划分函数 f。

算法 1. 划分函数 f

输入:算子 Op,设备子集 D

输出:算子内并行划分 $\mathcal{G}_{op}$

1. 初始化算子内并行划分:$\mathcal{G}_{op}=\{Op\}$
2. 初始化待划分算子集合:$\mathcal{G}_{\text{stage}}=\{Op\}$,设定初始划分维度 d
3. WHILE $\mathcal{G}_{\text{stage}}\neq$, DO
   a. 对于 $\mathcal{G}_{\text{stage}}$ 中的每个算子,沿维度 d 切分成 K 份,得到新的子算子集合
   $$\mathcal{G}_{\text{speit}}=\{\text{sub}Op_1,\text{sub}Op_2,\cdots,\text{sub}Op_K\}$$
   b. 用 $\mathcal{G}_{\text{speit}}$ 替换 $\mathcal{G}_{op}$ 中的 $\mathcal{G}_{\text{stage}}$
   c. 清空 $\mathcal{G}_{\text{stage}}$
   d. 确定下一个划分维度 d,将需要进一步划分的子算子添加到 $\mathcal{G}_{\text{stage}}$
4. 将 $\mathcal{G}_{op}$ 中的每个子算子映射到一台设备 $\text{Device}_k\in D$
5. 返回最终的算子内并行划分集合 $\mathcal{G}_{op}$

(3)流水线阶段划分 $\mathcal{G}_{\text{pipe}}$

对计算图 G 进行流水线阶段划分,得到计算子图集合 $\mathcal{G}_{\text{pipe}}=\{\text{sub}G_1,\text{sub}G_2,\cdots,\text{sub}G_Q\}$ 满足以下性质:

①连通性:每个子图 $\text{sub}G_i$ 包含的算子集合 $\mathcal{O}_i\subseteq\mathcal{O}$ 在 G 中诱导的子图是连通的。

②覆盖性:

$$\bigcup_{i=1}^{Q}\mathcal{O}_i=\mathcal{O}。$$

③不相交性:

$$\forall\, i\neq j,\quad \mathcal{O}_i\cap\mathcal{O}_i=\varnothing。$$

④设备与策略映射:每个子图 $\text{sub}G_i$ 映射到一个设备子集 $\mathcal{D}_i\subseteq\mathcal{D}$,并指定相应的算子内并行策略 A_{op}^{i}。

(4)流水线阶段划分策略 A_{pipe}

该策略描述如何对计算图进行流水线阶段划分:

①阶段数确定:决定将计算图划分为 Q 个阶段。

②划分位置选择:选择计算图中的划分点,将其分割为若干子图。

③设备映射:为每个子图 $\text{sub}G_i$ 分配设备子集 $\mathcal{D}_i$。

④算子内并行策略制定:为每个子图指定相应的算子内并行策略 A_{op}^{i}。

(5)流水线排布策略 $\mathcal{L}_{\text{pipe}}$

流水线排布策略 $\mathcal{L}_{\text{pipe}}$ 描述了流水化后整个计算过程的流水线排布:

$$\mathcal{L}_{\text{pipe}}=\{C_1,C_2,\cdots,C_R\}。$$

其中,每个约束 C_r 表示子图之间的依赖关系:

$$C_r:\text{sub}G_a\rightarrow\text{sub}G_b,$$

表示子图 $\text{sub}G_a$ 的计算必须在子图 $\text{sub}G_b$ 之前完成。这些约束确保了流水线的正确执行顺序。

(6)混合并行策略 A

综合以上,混合并行策略 A 包含两部分:流水线阶段划分策略 A_{pipe} 和流水线排布 $\mathcal{L}_{\text{pipe}}$,其中 A_{pipe} 又包含各子图的算子内并行策略 A_{op}^{i}:

$$A=(A_{\text{pipe}},\mathcal{L}_{\text{pipe}})=(A_{\text{pipe}}(\{A_{op}^{i}\}_{i=1}^{Q}),\mathcal{L}_{\text{pipe}})。$$

该策略全面描述了如何对计算图进行划分、映射和执行,以实现混合并行的目的。

根据上述定义,A 具有强大的表达能力,几乎可以表示所有目前已有的混合并行策略。当 A_{pipe} 划分阶段数为 1 时,表明不使用流水线并行变成一个纯算子内并行;当 A_{pipe} 给每个子图只映射一台设备时,则变成了一个纯粹的流水线并行不包含算子内并行。同时 $\mathcal{L}_{pp}$ 能表达流水线使用 1F1B,3F1B,Chinema 等等排布方式。

5.2 基于新范式的混合并行扩展策略空间

目前,即便是最先进的混合并行方法所能组合出的策略在灵活性方面仍显不足,尤其是在异构计算环境下。例如,当前主流的混合并行方法通常将流水线并行与算子内并行相结合。以一种常见的混合并行策略表达方式 $TP_a-DP_b-PP_c$ 为例,根据本文的定义,这相当于将模型划分为 c 个流水线阶段,每个阶段内的每个子部分 sub$G$ 映射到 $a\times b$ 台设备的子集上进行算子内并行。并且每个子部分的并行操作顺序为先张量并行后数据并行。在完成张量并行划分后,对所有部分统一进行数据并行划分。尽管拿目前相对先进的 alpa 策略来说,虽然他能同时解决 A_{pipe} 和 A_{op} 的划分,但是其在 A_{op} 的划分上也加了沿各维度均等划分的约束。

然而,这种混合并行策略的划分方式往往预先隐含了诸多约束,例如:

(1)每个 sub$G$ 通常映射相同数量、相同类型的设备,甚至采用相同的 A_{op}。

(2)每个 $subG$ 所映射的设备集群 D 通常没有交集,也就是说每台设备往往只专注于计算一块计算子图 $subG$ 。

目前一些手工设计的新方案已经部分突破了这些限制。例如,对于 T5 模型[125],由于 Embedding 层过大,该算子会被映射到所有设备上进行计算。Megatron-LM 设计了交错流水线[126],每相隔一定数量的 $subG$ 会映射到同一个设备集群 D ,并支持不同算子使用不同的 TP 维度。这些新设计通常通过手动调整部分策略来解除混合并行策略的部分约束,从而获得了显著的性能提升。然而,实际上仍有许多隐含的约束尚未被解除,这限制了策略空间的进一步优化拓展。

例如,当我们放宽各个 $subG_i$ 必须采用相同算子内并行策略的约束时,可以推导出一种优化后的混合并行策略扩展方案。由于异构设备之间的计算性能存在差异,可能会使得 $subG_1$ 采用 m 台 A 型设备进行 TP 并行,而 $subG_2$ 则利用 n 台 B 型设备进行 DP 并行。这样的策略灵活地协调了流水线中各子图的计算时长,从而实现了更优的流水线排布效率。

再如,当我们放宽算子内并行策略必须形成笛卡尔积空间的限制时,可以获得另一种混合并行策略扩展解。以某个 $subG$ 映射到两台 A 型设备和一台 B 型设备为例,现有的自动并行搜索策略通常只能沿某一维度将其划分为三部分。然而,在异构场景下,设备 A 与设备 B 之间的通信性能可能难以满足 TP 和 CP 的要求,而采用纯 DP3 划分则会导致模型存储多份参数,从而浪费显存。基于本文提出的策略定义,可以先沿 DP 维度将 $subG$ 划分为两部分:将其中一部分分配给设备 B,另一部分再沿 TP 维度细分为两份,分配给设备 A。这样的策略在同构场景下或许显得冗余,但在异构环境中则是一种切实且必要的优化手段。

总体而言,现有的混合并行方案在处理异构环境时仍存在诸多局限,主要体现在策略的预设约束和灵活性不足上。基于上文的混合并行新范式,我们能扩展出更为灵活的混合并行策略空间,允许在异构设备上进行更复杂的划分和映射。通过解除现有方案中的隐含约束,我们可以实现更高效的流水线排布和算子内并行策略,从而提升整体计算性能。

5.3　基于新范式的自动并行

由于混合并行策略繁多,且每个策略维度都有相当多的划分粒度,自动混合并行是一个 NP 难的问题。基于上文的定义,自动混合并行往往需要解决三个问题:

(1)确定计算图的划分与映射 A_{pipe} ,

(2)制定算子内并行策略 A_{op} ,

(3)流水化与流水线重排 $\mathcal{L}_{\text{pipe}}$ 。

现有的自动并行方案往往只挑选部分问题进行解决,同时会先验性地增加一系列约束条件限制解空间。比如 vPipe[101],不考虑算子内并行策略,将流水线排布确定为 1F1B,只讨论计算图的划分策略。

目前大部分自动并行策略往往都将 $\mathcal{L}_{\text{pipe}}$ 固定,比如限制为 1F1B 流水线,或者干脆不讨论 $\mathcal{L}_{\text{pipe}}$,仅给出 $\mathcal{L}_{\text{pipe}}$ 的约束为尽可能均等化每个计算子图的计算时间。但也有一些工作在讨论有关流水线重排的问题,比如 Tessel[127],其通过灵活的调度搜索机制,自动寻找适合不同操作符放置策略的高效流水线。Tessel 的关键创新在于利用重复执行模式(repetend)的发现,将复杂的调度问题分解为较小规模的搜索空间,从而在不牺牲性能的情况下,大幅减少调度的搜索时间。相比于传统的预定义调度方式,Tessel 不仅能够优化经典的 1F1B 调度,还能够探索出在多种模型和计算需求下更加灵活且高效的调度方案。

6　后续研究方向

6.1　面向异构设备的训练系统优化

针对当前国内异构计算环境的特点,需重点突破异构设备间的协同优化问题。首先需设计高效的异构通信协议,解决国产 GPU 与不同架构 AI 芯片之间的跨设备通信瓶颈,特别是处理 PCIe/NVLink 异构互联场景下的带宽不对称问题[128];其次应构建大规模异构集群拓扑感知系统,建立基于 RDMA/RoCE 的多级通信拓扑建模,并开发面向 NUMA 架构的通信优化算法[129-130]。在算子优化层面,需建立异构设备能力画像系统,通过动态性能分析确定各设备擅长的算子类型,并设计基于设备特性的算子自动分配器[131-132]。此外,还需解决国产芯片与 NVIDIA GPU 在计算精度、指令集等方面的差异问题,构建统一的量化训练框架[133]。

6.2　更灵活的混合并行策略制定

如 4.3 节所述,现有的混合并行策略在灵活性方面存在不足,尤其在异构计算环境下表现尤为明显。未来的研究应致力于制定更加灵活的策略,充分考虑异构计算设备的计算和通信特性。例如,可以引入动态策略生成方法,根据实际的设备性能和任务负载自

适应地调整并行策略，而不再拘泥于固定的张量并行或数据并行组合。这样的灵活性有助于提高系统对复杂环境的适应能力，最大化并行效率。

6.3 面向异构集群的并行策略搜索

大部分现有研究都是基于同构集群进行的，这些设备由于计算和通信能力相近，策略制定和搜索过程中可以通过先验约束缩小解空间。然而，异构集群中的设备计算能力和通信能力存在差异，传统的约束条件将不再适用，导致解空间进一步扩大。因此，未来的研究需要针对异构集群开发新的策略搜索算法，充分考虑设备拓扑结构、通信带宽、计算能力等硬件特性。这将为提高异构集群的训练效率和并行策略的适应性提供更好的解决方案。

6.4 更全面的自动混合并行策略搜索

目前的自动混合并行方案由于问题复杂性而产生庞大的解空间，通常仅针对第四章中提到的部分问题进行讨论。这限制了混合并行策略的优化，无法实现全局最优解。未来的研究可以尝试将 A_{pipe}、A_{op} 和 $\mathcal{L}_{\text{pipe}}$ 三方面的因素结合在一起进行联合优化。例如，当前的 A_{pipe} 划分策略常常受限于 $\mathcal{L}_{\text{pipe}}$ 的先验约束，导致代价模型被设定为尽可能均衡计算子图的计算和通信时长。然而，如果我们能够打破这一先验限制，通过重排 $\mathcal{L}_{\text{pipe}}$，可能会发现这种约束并非必要，进而扩大搜索的优化空间，从而寻找到更具全局优化潜力的策略。

参 考 文 献

[1] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature, 2015, 521(7553): 436-444

[2] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks//Proceedings of the Advances in Neural Information Processing Systems. Lake Tahoe, USA, 2012: 84-90

[3] Liu Y, Han T, Ma S, et al. Summary of ChatGPT-related research and perspective towards the future of large language models. Meta-radiology, 2023, 1(2): 100017

[4] Brown T, Mann B, Ryder N, et al. Language models are few-shot learners//Proceedings of the Advances in Neural Information Processing Systems. Virtual, 2020: 1877-1901

[5] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need//Proceedings of the Advances in Neural Information Processing Systems. Long Beach, USA, 2017: 6000-6010

[6] Dosovitskiy A, Beyer L, Kolesnikov A, et al. An image is worth 16x16 words: Transformers for image recognition at scale//Proceedings of the International Conference on Learning Representations. Virtual, Austria, 2020: 611-632

[7] Kaplan J, McCandlish S, Henighan T, et al. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361, 2020

[8] Tripp C E, Perr-Sauer J, Gafur J, et al. Measuring the energy consumption and efficiency of deep neural networks: An empirical analysis and design recommendations. arXiv preprint arXiv:2403.08151, 2024

[9] Devlin J, Chang M-W, Lee K, et al. BERT: Pre-training of deep bidirectional transformers for language understanding//Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Minneapolis, Minnesota, 2019: 4171-4186

[10] Anil R, Dai A M, Firat O, et al. PaLM 2 technical report. arXiv preprint arXiv:2305.10403, 2023

[11] Strubell E, Ganesh A, McCallum A. Energy and policy considerations for deep learning in NLP//Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. Florence, Italy, 2019: 3645-3650

[12] Schwartz R, Dodge J, Smith N A, et al. Green AI. arXiv preprint arXiv:1907.10597, 2019

[13] OpenAI, Achiam J, Adler S, et al. GPT-4 technical report. arXiv preprint, 2024

[14] Shazeer N, Mirhoseini A, Maziarz K, et al. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer//Proceedings of the International Conference on Learning Representations. Toulon, France, 2017: 878-897

[15] Touvron H, et al. LLaMA: Open and efficient foundation language models. Arxiv Preprint Arxiv:2302.13971, 2023

[16] Paszke A, et al. PyTorch: An imperative style, high-performance deep learning library. Arxiv Preprint Arxiv: 1912.01703, 2019

[17] Abadi M, et al. TensorFlow: a system for large-scale machine learning. Arxiv Preprint Arxiv:1605.08695, 2016

[18] Rasley J, Rajbhandari S, Ruwase O, et al. DeepSpeed: system optimizations enable training deep learning models with over 100 billion parameters//Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. New York, USA, 2020: 3505-3506

[19] Rajbhandari S, Rasley J, Ruwase O, et al. ZeRO: Memory optimizations toward training trillion parameter models//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Atlanta, USA, 2020: 1-16

[20] DeepSeek-AI, Liu A, Feng B, et al. DeepSeek-V3 Technical Report. arXiv preprint arXiv:2412.19437, 2024

[21] Grattafiori A, Dubey A, Jauhri A, et al. The llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024

[22] Langer M, He Z, Rahayu W, et al. Distributed training of deep learning models: A taxonomic perspective. IEEE Trans. Parallel Distrib. Syst., 2020, 31(12): 2802-2818

[23] Liang P, Tang Y, Zhang X, et al. A survey on auto-parallelism of large-scale deep learning training. IEEE Transactions on Parallel and Distributed Systems, 2023, 34(8): 2377-2390

[24] Zhang Z-X, Wen Y-B, Lyu H-Q, et al. AI computing systems for large language models training. Journal of Computer Science and Technology, 2025, 40(1): 6-41

[25] Mayer R, Jacobsen H-A. Scalable deep learning on distributed infrastructures: challenges, techniques, and tools. ACM Computing Surveys, 2021, 53(1): 1-37

[26] Verbraeken J, Wolting M, Katzy J, et al. A survey on distributed machine learning. ACM Computing Surveys, 2020, 53(2): 30:1-30:33

[27] Shallue C J, Lee J, Antognini J, et al. Measuring the effects of data parallelism on neural network training. Journal of Machine Learning Research, 2019, 20(112): 1-49

[28] Chen C-C, Yang C-L, Cheng H-Y. Efficient and robust parallel DNN training through model parallelism on multi-GPU platform. Arxiv E-prints, 2018: arXiv:1809.2839

[29] Shoeybi M, Others. Megatron-LM: training multi-billion parameter language models using model parallelism. Arxiv Preprint Arxiv:1909.08053, 2020

[30] Narayanan D, Phanishayee A, Shi K, et al. Memory-efficient pipeline-parallel DNN training//Proceedings of the 38th International Conference on Machine Learning. Virtual, 2021: 7937-7947

[31] Li S, Xue F, Baranwal C, et al. Sequence parallelism: Long sequence training from system perspective//Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics. Toronto, Canada, 2023: 2391-2404

[32] Cai W, Jiang J, Qin L, et al. Shortcut-connected expert parallelism for accelerating mixture-of-experts. Arxiv E-prints, 2024: arXiv:2404.5019

[33] Singh S, Ruwase O, Awan A A, et al. A hybrid tensor-expert-data parallelism approach to optimize mixture-of-experts training//Proceedings of the 37th ACM International Conference on Supercomputing. New York, USA, 2023: 203-214

[34] Lai Z, Li S, Tang X, et al. Merak: An efficient distributed DNN training framework with automated 3D parallelism for giant foundation models. IEEE Transactions on Parallel and Distributed Systems, 2023, 34(5): 1466-1478

[35] Bai J, Bai S, Chu Y, et al. Qwen technical report. arXiv preprint arXiv:2309.16609, 2023

[36] Glm T, Zeng A, et al. ChatGLM: A family of large language models from GLM-130B to GLM-4 all tools. 2024

[37] Rumelhart D E, Hinton G E, Williams R J. Learning representations by back-propagating errors. Nature, 1986, 323(6088): 533-536

[38] Zheng L, Li Z, Zhang H, et al. Alpa: Automating inter- and intra-operator parallelism for distributed deep learning//Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation. Carlsbad, USA, 2022: 559-578

[39] Korthikanti V, Others. Reducing activation recomputation in large transformer models. arxiv preprint arxiv: 2205.05198, 2022

[40] Huang Y, Cheng Y, Bapna A, et al. GPipe: efficient training of giant neural networks using pipeline parallelism//Proceedings of the Advances in Neural Information Processing Systems. Vancouver, Canada, 2019: 103-112

[41] Narayanan D, Harlap A, Phanishayee A, et al. PipeDream: Generalized pipeline parallelism for DNN training//Proceedings of the 27th ACM Symposium on Operating Systems Principles. Huntsville, Canada, 2019: 1-15

[42] Fan S, Rong Y, Meng C, et al. DAPPLE: A pipelined data parallel approach for training large models//Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York, USA, 2021: 431-445

[43] Qi P, Wan X, Huang G, et al. Zero bubble (almost) pipeline parallelism//Proceedings of the Twelfth International Conference on Learning Representations. Kigali, Rwanda, 2023: 47619-47635

[44] Li Z, Zhuang S, Guo S, et al. TeraPipe: Token-level pipeline parallelism for training large-scale language models//Proceedings of the 38th International Conference on Machine Learning. Virtual Event, 2021: 6543-6552

[45] Liu H, Abbeel P. Blockwise parallel transformers for large context models//Proceedings of the Thirty-seventh Conference on Neural Information Processing Systems. New Orleans, USA, 2023: 8828-8844

[46] Dao T, Fu D, Ermon S, et al. FlashAttention: Fast and memory-efficient exact attention with IO-awareness//Proceedings of the Advances in Neural Information Processing Systems. New Orleans, USA, 2022: 16344-16359

[47] Dao T. FlashAttention-2: faster attention with better parallelism and work partitioning. arXiv preprint, 2023

[48] Milakov M, Gimelshein N. Online normalizer calculation for softmax. 2018

[49] Liu H, Zaharia M, Abbeel P. RingAttention with blockwise transformers for near-infinite context//Proceedings of the Twelfth International Conference on Learning Representations. Vienna, Austria, 2023: 56456-56473

[50] Jacobs S A, Tanaka M, Zhang C, et al. DeepSpeed ulysses: System optimizations for enabling training of extreme long sequence transformer models. Arxiv Preprint Arxiv:2309.14509, 2023

[51] Ainslie J, Lee-Thorp J, de Jong M, et al. GQA: Training generalized multi-query transformer models from multi-head checkpoints//Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. Singapore, 2023: 4895-4901

[52] Fang J, Zhao S. USP: a unified sequence parallelism ap-

proach for long context generative AI. arXiv preprint arXiv: 2405.07719, 2024

[53] Hwang C, Cui W, Xiong Y, et al. Tutel: Adaptive mixture-of-experts at scale//Proceedings of the Machine Learning and Systems. Miami, USA, 2023: 269-287

[54] Fedus W, Zoph B, Shazeer N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. Journal of Machine Learning Research, 2022, 23(120): 1-39

[55] Lepikhin D, Lee H, Xu Y, et al. GShard: Scaling giant models with conditional computation and automatic sharding//Proceedings of the International Conference on Learning Representations. Virtual , Austria, 2020: 14225-14248

[56] He J, Qiu J, Zeng A, et al. FastMoE: A fast mixture-of-expert training system. arXiv preprint arXiv:2103.13262, 2021

[57] Gale T, Narayanan D, Young C, et al. MegaBlocks: Efficient sparse training with mixture-of-experts//Proceedings of the Machine Learning and Systems. Miami, USA, 2023: 288-304

[58] Jacobs R A, Jordan M I, Nowlan S J, et al. Adaptive mixtures of local experts. Neural Computation, 1991, 3(1): 79-87

[59] Rajbhandari S, Li C, Yao Z, et al. DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale//Proceedings of the 39th International Conference on Machine Learning. Baltimore, USA, 2022: 18332-18346

[60] Zhao C, Zhou S, Zhang L, et al. DeepEP: An efficient expert-parallel communication library. GitHub, 2025

[61] Qiu Z, Huang Z, Zheng B, et al. Demons in the detail: On implementing load balancing loss for training specialized mixture-of-expert models//Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics. Vienna, Austria, 2025: 5005-5018

[62] Dean J, Corrado G, Monga R, et al. Large scale distributed deep networks//Proceedings of the Advances in Neural Information Processing Systems. Lake Tahoe, USA, 2012: 1223-1231

[63] Jain A, Awan A A, Aljuhani A M, et al. GEMS: GPU-enabled memory-aware model-parallelism system for distributed DNN training//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Virtual, 2020: 1-15

[64] Li S, Hoefler T. Chimera: Efficiently training large-scale neural networks with bidirectional pipelines//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. St. Louis, USA, 2021: 1-14

[65] Wan X, Qi P, Huang G, et al. PipeOffload: Improving scalability of pipeline parallelism with memory optimization//Proceedings of the Forty-second International Conference on Machine Learning. Vancouver, Canada, 2025: 61942-61955

[66] Chang L-W, Bao W, Hou Q, et al. FLUX: Fast software-based communication overlap on gpus through kernel fusion. arXiv preprint arXiv:2406.06858, 2024

[67] Tu J, Zhou J, Ren D. An asynchronous distributed training algorithm based on gossip communication and stochastic gradient descent. Computer Communications, 2022, 195: 416-423

[68] Jia X, Song S, He W, et al. Highly scalable deep learning training system with mixed-precision: training ImageNet in four minutes. 2018

[69] Wang G, View Profile, Lei Y, et al. Communication-efficient ADMM-based distributed algorithms for sparse training. Neurocomputing, 2023, 550(C):126456

[70] Devraj A, Ding E, Kumar A V, et al. Accelerating AllReduce with a persistent straggler. arXiv preprint arXiv: 2505.23523, 2025

[71] Xiong D, Chen L, Jiang Y, et al. Revisiting the time cost model of AllReduce. 2024

[72] Wei Y, Liu M, Wu W. AllReduce scheduling with hierarchical deep reinforcement learning. arXiv preprint arXiv: 2503.21013, 2025

[73] Alistarh D, Grubic D, Li J, et al. QSGD: Communication-efficient SGD via gradient quantization and encoding//Proceedings of the Advances in Neural Information Processing Systems. Long Beach, USA, 2017: 1707-1718

[74] Bernstein J, Wang Y-X, Azizzadenesheli K, et al. signSGD: Compressed optimisation for non-convex problems//Proceedings of the 35th International Conference on Machine Learning. Stockholm, Sweden, 2018: 560-569

[75] Lin Y, Han S, Mao H, et al. Deep gradient compression: Reducing the communication bandwidth for distributed training. arXiv preprint arXiv:1712.01887,2020

[76] Stich S U, Cordonnier J-B, Jaggi M. Sparsified SGD with memory//Proceedings of the Advances in Neural Information Processing Systems. Montréal, Canada, 2018: 4452-4463

[77] Vogels T, Karimireddy S P, Jaggi M. PowerSGD: Practical low-rank gradient compression for distributed optimization//Proceedings of the Advances in Neural Information Processing Systems. Vancouver, Canada, 2019: 14269-14278

[78] Zhang L, Zhang L, Shi S, et al. Evaluation and optimization of gradient compression for distributed deep learning//Proceedings of the 2023 IEEE 43rd International Conference on Distributed Computing Systems. Hong Kong, China, 2023: 361-371

[79] Yoon D, Oh S. MiCRO: Near-zero cost gradient sparsification for scaling and accelerating distributed DNN training//Proceedings of the 2023 IEEE 30th International Conference on High Performance Computing, Data, and Analytics. Goa, India, 2023: 87-96

[80] Polyakov I, Dukhanov A, Spirin E. TAGC: Optimizing gradient communication in distributed transformer training//

Proceedings of the 5th Workshop on Machine Learning and Systems. Rotterdam, The Netherlands, 2025: 254-260

[81] Markov I, Vladu A, Guo Q, et al. Quantized distributed training of large models with convergence guarantees//Proceedings of the 40th International Conference on Machine Learning. Honolulu, USA, 2023: 24020-24044

[82] Jangda A, Huang J, Liu G, et al. Breaking the computation and communication abstraction barrier in distributed machine learning workloads. arXiv preprint arXiv:2105.05720, 2022

[83] Harlap A, Narayanan D, Phanishayee A, et al. PipeDream: Fast and efficient pipeline parallel DNN training. arXiv preprint arXiv:1806.03377, 2018

[84] Wu K, Park J B, Zhang X, et al. SSDTrain: An activation offloading framework to SSDs for faster large language model training. Arxiv Preprint Arxiv:2408.10013, 2024

[85] Yan R, Jiang Y, Tao W, et al. FlashFlex: Accommodating large language model training over heterogeneous environment. Arxiv Preprint Arxiv:2409.01143, 2024

[86] Ren J, Rajbhandari S, Aminabadi R Y, et al. {ZeRO-offload}: democratizing {billion-scale} model training//Proceedings of the 2021 USENIX Annual Technical Conference. Santa Clara, USA, 2021: 551-564

[87] Yao J, Jacobs S A, Tanaka M, et al. Training ultra long context language model with fully pipelined distributed transformer. arXiv preprint arXiv:2408.16978, 2024

[88] Griewank A, Walther A. Algorithm 799: Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. ACM Transactions on Mathematical Software, 2000, 26(1): 19-45

[89] Chen T, Xu B, Zhang C, et al. Training deep nets with sublinear memory cost. arXiv preprint arXiv:1604.06174, 2016

[90] Douillard A, Feng Q, Rusu A A, et al. DiLoCo: distributed low-communication training of language models. arXiv preprint arXiv:2311.08105, 2024

[91] Ho Q, Cipar J, Cui H, et al. More effective distributed ML via a stale synchronous parallel parameter server//Proceedings of the Advances in Neural Information Processing Systems. Lake Tahoe, USA, 2013: 1223-1231

[92] Nabli A, Fournier L, Erbacher P, et al. ACCO: Accumulate while you communicate, hiding communications in distributed LLM training//Proceedings of the OPT 2024 Optimization for Machine Learning. San Diego, USA, 2024

[93] FlagOpen. FlagScale: A large model toolkit based on open-sourced projects. 2025

[94] Kwon W, Li Z, Zhuang S, et al. Efficient memory management for large language model serving with PagedAttention//Proceedings of the 29th Symposium on Operating Systems Principles. New York, USA, 2023: 611-626

[95] Li S, Liu H, Bian Z, et al. Colossal-AI: A unified deep learning system for large-scale parallel training//Proceedings of the 52nd International Conference on Parallel Processing. New York, USA, 2023: 766-775

[96] Chen Q, Gu D, Wang G, et al. InternEvo: Efficient long-sequence large language model training via hybrid parallelism and redundant sharding. arXiv preprint arXiv:2401.09149, 2024

[97] Zhang W, Zhou B, Tang X, et al. MixPipe: Efficient bidirectional pipeline parallelism for training large-scale models//Proceedings of the 2023 60th ACM/IEEE Design Automation Conference. San Francisco, USA, 2023: 1-6

[98] Jia Z, Zaharia M, Aiken A. Beyond data and model parallelism for deep neural networks//Proceedings of the Machine Learning and Systems. California, USA, 2019: 1-13

[99] Shazeer N, Cheng Y, Parmar N, et al. Mesh-tensorFlow: Deep learning for supercomputers//Proceedings of the Advances in Neural Information Processing Systems. Montréal, Canada, 2018: 10435-10444

[100] Aminabadi R Y, Others. DeepSpeed inference: Enabling efficient inference of transformer models at unprecedented scale. arxiv preprint arxiv:2207.00032, 2022

[101] Zhao S, Li F, Chen X, et al. vPipe: A virtualized acceleration system for achieving efficient and scalable pipeline parallel DNN training. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(3): 489-506

[102] Sun Z, Cao H, Wang Y, et al. AdaPipe: Optimizing pipeline parallelism with adaptive recomputation and partitioning//Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA, 2024: 86-100

[103] KennedyKen, KremerUlrich. Automatic data layout for distributed-memory machines. ACM Transactions on Programming Languages and Systems, 1998, 20(4): 869-916

[104] Li J, Chen M. Index domain alignment: Minimizing cost of cross-referencing between distributed arrays//Proceedings of the Third Symposium on the Frontiers of Massively Parallel Computation. Maryland, USA, 1990: 424-433

[105] Kremer U. NP-completeness of dynamic remapping//Proceedings of the Fourth Workshop on Compilers for Parallel Computers. Delft, The Netherlands, 1993: 1-7

[106] Li J, Chen M. The data alignment phase in compiling programs for distributed-memory machines. Journal of Parallel and Distributed Computing, 1991, 13(2): 213-221

[107] Um T, Oh B, Kang M, et al. Metis: Fast automatic distributed training on heterogeneous {GPUs}//Proceedings of the 2024 USENIX Annual Technical Conference. Santa Clara, USA, 2024: 563-578

[108] Zhou Y, Roy S, Abdolrashidi A, et al. Gdp: Generalized device placement for dataflow graphs. Arxiv Preprint Arxiv:1910.01578, 2019

[109] Lu W, Yan G, Li J, et al. Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks//Proceedings of the 2017 IEEE International Symposium on

High Performance Computer Architecture. Austin, USA, 2017: 553-564

[110] Schaarschmidt M, Grewe D, Vytiniotis D, et al. Automap: Towards ergonomic automated parallelism for ML models. Arxiv Preprint Arxiv:2112.02958, 2021

[111] Mirhoseini A, Goldie A, Pham H, et al. A hierarchical model for device placement//Proceedings of the International Conference on Learning Representations. Vancouver, Canada, 2018: 2261-2272

[112] Gilks W R, Richardson S, Spiegelhalter D. Markov chain monte carlo in practice. New York, USA: Chapman and Hall/CRC, 1995

[113] Lattner C, Amini M, Bondhugula U, et al. MLIR: Scaling compiler infrastructure for domain specific computation//Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization. Seoul, Republic of Korea, 2021: 2-14

[114] Jia Z, Lin S, Qi C R, et al. Exploring hidden dimensions in accelerating convolutional neural networks//Proceedings of the International Conference on Machine Learning. Stockholm, Sweden, 2018: 2274-2283

[115] Tarnawski J M, Narayanan D, Phanishayee A. Piper: Multidimensional planner for dnn parallelization. Advances in Neural Information Processing Systems, 2021, 34: 24829-24840

[116] Song L, Chen F, Zhuo Y, et al. AccPar: Tensor partitioning for heterogeneous deep learning accelerators//Proceedings of the 2020 IEEE International Symposium on High Performance Computer Architecture. San Diego, USA, 2020: 342-355

[117] Cai Z, Yan X, Ma K, et al. TensorOpt: Exploring the tradeoffs in distributed DNN training with auto-parallelism. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(8): 1967-1981

[118] Liu X, Wang Y, Zhu S, et al. Galvatron: An automatic distributed system for efficient foundation model training. arXiv preprint, 2025

[119] Miao X, Wang Y, Jiang Y, et al. Galvatron: Efficient transformer training over multiple GPUs using automatic parallelism. Proceedings of the VLDB Endowment, 2022, 16(3): 470-479

[120] Wang Y, Jiang Y, Miao X, et al. Improving automatic parallel training via balanced memory workload optimization. IEEE Transactions on Knowledge and Data Engineering, 2024, 36(8): 3906-3920

[121] She R, Pang B, Li K, et al. Automatic operator-level parallelism planning for distributed deep learning-a mixed-integer programming approach. arXiv preprint arXiv:2503.09357, 2025

[122] Zhu Z, Giannoula C, Andoorveedu M, et al. Mist: Efficient distributed training of large language models via memory-parallelism Co-optimization//Proceedings of the Twentieth European Conference on Computer Systems. Rotterdam, the Netherlands, 2025: 1298-1316

[123] Lin H, Wu K, Li J, et al. UniAP: Unifying inter- and intra-layer automatic parallelism by mixed integer quadratic programming//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. Tennessee, USA, 2025: 20947-20957

[124] Liu J, Wu Z, Feng D, et al. HeterPS: Distributed deep learning with reinforcement learning based scheduling in heterogeneous environments. Future Generation Computer Systems, 2023, 148(C): 106-117

[125] Raffel C, Shazeer N, Roberts A, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. The Journal of Machine Learning Research, 2020, 21(1): 140:5485-140:5551

[126] Narayanan D, Others. Efficient large-scale language model training on GPU clusters using megatron-LM. Arxiv Preprint Arxiv:2104.04473, 2021

[127] Lin Z, Miao Y, Xu G, et al. Tessel: Boosting distributed execution of large DNN models via flexible schedule search//Proceedings of the 2024 IEEE International Symposium on High-Performance Computer Architecture. Edinburgh, UK, 2024: 803-816

[128] Li A, Song S L, Chen J, et al. Evaluating modern GPU interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(1): 94-110

[129] Gangidi A, Miao R, Zheng S, et al. RDMA over ethernet for distributed training at meta scale//Proceedings of the ACM SIGCOMM 2024 Conference. New York, USA, 2024: 57-70

[130] Li Q, Gao Y, Wang X, et al. Flor: An open high performance {RDMA} framework over heterogeneous {RNICs}//Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation. Boston, USA, 2023: 931-948

[131] Amine Hamdi M, Daghero F, Maria Sarda G, et al. MATCH: Model-aware TVM-based compilation for heterogeneous edge devices. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2025, 44(10): 3844-3857

[132] Zheng B, Jiang Z, Yu C H, et al. DietCode: Automatic optimization for dynamic tensor programs//Proceedings of the Machine Learning and Systems. Santa Clara, USA, 2022: 848-863

[133] Zhao J, Wan B, Peng Y, et al. QSync: Quantization-minimized synchronous distributed training across hybrid devices//Proceedings of the 2024 IEEE International Parallel and Distributed Processing Symposium. San Francisco, USA, 2024: 193-204

ZHANG Gui-Peng, Ph. D. candidate. His main research interests include cloud gaming, distributed training and heterogeneous computing.

SUN Yu-Zhong, Ph. D., professor. His main research interests include big data intelligence analysis (machine learning) and calculation.

Background

The research in this paper addresses the field of distributed systems for artificial intelligence, specifically focusing on the challenge of training large-scale models based on the Transformer architecture. With the exponential growth in model parameters, reaching hundreds of billions or even trillions, traditional single-mode parallel methods like data parallelism have hit a bottleneck. They face severe limitations in computational efficiency, memory capacity, and communication overhead. Consequently, hybrid parallelism, which combines multiple strategies such as data, tensor, sequence, pipeline, and expert parallelism, has emerged as the dominant paradigm to tackle these challenges.

Internationally, the state-of-the-art is characterized by powerful distributed training frameworks like NVIDIA's Megatron-LM and Microsoft's DeepSpeed. These systems have successfully enabled the training of massive models by manually combining various parallel techniques. However, the current understanding and application of hybrid parallelism remain somewhat fragmented and reliant on empirical, case-by-case engineering. There is a notable lack of a unified theoretical model to systematically analyze the intricate coupling relationships between different strategies and define their combination boundaries. This gap makes it difficult to explore the vast design space of parallelism strategies in a principled manner.

This survey paper addresses this theoretical gap by establishing a unified framework for hybrid parallelism. It deconstructs existing parallel strategies from the fundamental concepts of intra-operator and inter-operator partitioning, providing a coherent mathematical representation. This framework serves as an extensible analytical tool to systematically understand, compare, and design complex hybrid parallel configurations. Furthermore, this paper charts the evolutionary path of automatic parallelism search techniques, which aim to automate the discovery of optimal training strategies, moving beyond manual tuning.

This work is a fundamental part of the Science and Technology Innovation 2030—Major Project "Elastic Accelerated Distributed Training with Multi-core and Multi-engine" (No. 2022ZD0119104). The project's primary significance lies in developing the core technologies required to break through the systemic challenges of training future trillion-parameter models, which is crucial for advancing generative AI and maintaining competitiveness in this strategic field. Our research group has a consistent track record in distributed systems, cloud computing, and heterogeneous computing. This survey builds upon that expertise to provide the theoretical foundation for the project. Specifically, the contributions of this paper—the unified framework and the systematic analysis—form the essential theoretical groundwork that will guide the project's subsequent research and development of novel, efficient, and automated distributed training systems.