中国科学: 信息科学 2023 年 第 53 卷 第 8 期: 1441–1468
SCIENTIA SINICA Informationis
《中国科学》杂志社
SCIENCE CHINA PRESS
国防科技大学建校 70 周年专题 • 评述

并行智能训练技术: 挑战与发展

卢凯*, 赖志权, 李笙维, 柳炜杰, 葛可适, 卢锡城, 李东升

国防科技大学计算机学院并行与分布处理重点实验室, 长沙 410073
* 通信作者. E-mail: kailu@nudt.edu.cn

收稿日期: 2023–02–27; 修回日期: 2023–05–04; 接受日期: 2023–06–15; 网络出版日期: 2023–08–17

国家自然科学基金 (批准号: 62025208) 和国家重点研发计划 (批准号: 2021YFB0301200) 资助项目

摘要 近年来, 以深度学习为代表的人工智能技术迅猛发展, 深度学习模型和训练数据的规模均呈爆炸式增长, 给智能模型训练系统带来了巨大挑战. 随着高性能计算与人工智能的不断深度融合, 并行智能训练技术成为大规模深度学习模型高效训练的主要方法. 本文总结了并行智能训练的基本模式和关键技术, 以及并行智能训练框架的发展现状, 分析了并行智能训练技术和框架发展面临的挑战与发展趋势, 简介了银河天璇并行智能训练框架的研究进展.

关键词 智能训练, 高性能计算, 并行智能训练, 深度学习

1 引言

海量数据的获得以及智能算力的提升推动人工智能迎来第三次发展浪潮. 2006 年以来, 深度学习技术飞速发展, 在计算机视觉、自然语言处理、科学发现等领域取得了广泛的应用. 2022 年底发布的 ChatGPT 对话模型[1)] 进一步引发了人工智能热潮.

伴随人工智能技术的快速发展, 智能模型参数和训练数据的规模呈爆炸式增长. 2017 年, Facebook 使用 256 块 P100 GPU 在一个小时内完成了 ResNet50 [1] 在 ImageNet-1K 数据集 [2] 上的训练, ResNet50 模型的参数量为 2500 万, ImageNet-1K 训练数据集包含 128 万张图像. 到 2020 年, OpenAI 使用 10000 块 V100 GPU 完成 GPT-3 模型 [3] 的训练, GPT-3 模型的参数量为 1750 亿, 训练数据使用的文本语料大小为 570 GB (Gigabyte), 包含 4000 亿个单词 (tokens), 训练的计算量达到 3.14E+23 浮点运算次数 (floating point operations, FLOPs), 训练时间数以月计. 可以说, 人工智能飞速发展离不开大模型、大数据、大算力的 "暴力美学".

人工智能训练的超大规模算力需求, 使其逐渐成为一种新型的高性能计算应用, 促进了人工智能与高性能计算的不断融合发展. 2017 年, 优步公司 (Uber) 的 Horovod [4] 数据并行训练框架, 通过借

1) ChatGPT. OpenAI Blog, 2022. A15–A24. https://openai.com/blog/chatgpt/.

引用格式: 卢凯, 赖志权, 李笙维, 等. 并行智能训练技术: 挑战与发展. 中国科学: 信息科学, 2023, 53: 1441–1468, doi: 10.1360/SSI-2023-0051
Lu K, Lai Z Q, Li S W, et al. Parallel intelligent computing: development and challenges (in Chinese). Sci Sin Inform, 2023, 53: 1441–1468, doi: 10.1360/SSI-2023-0051

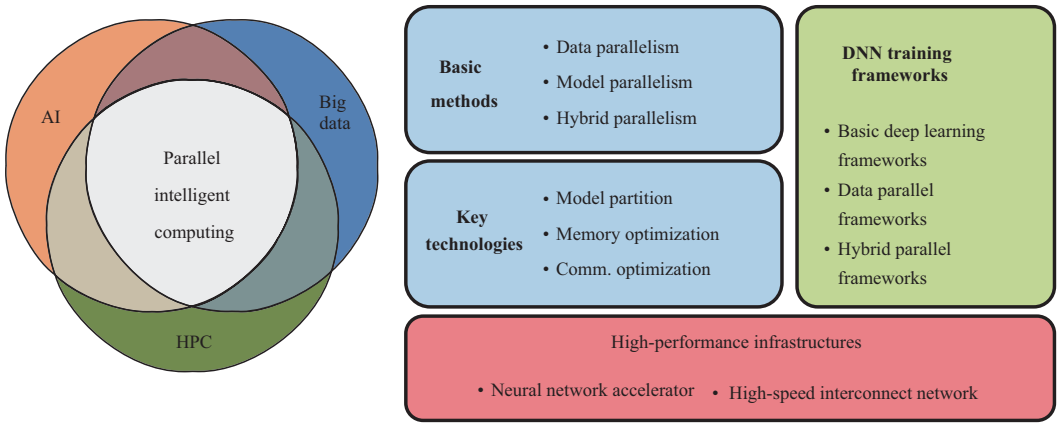

图 1 (网络版彩图) 并行智能训练的基本概念和技术体系. (a) 并行智能训练是人工智能、大数据、高性能计算的交叉领域和 (b) 并行智能训练技术体系

Figure 1 (Color online) Concept and technologies of parallel deep learning. (a) Parallel deep learning is at the intersection between AI, big data, and HPC and (b) parallel deep learning technology stack

鉴高性能计算中的 Ring AllReduce[2)] 技术, 尝试解决传统参数服务器 [5] (parameter server) 分布式训练架构中参数聚合通信的可扩展性问题, 标志着人工智能和高性能计算技术融合发展的开端. 随着 BERT [6], GPT [7] 等基于 Transformer [8] 的预训练语言模型的发展, 智能模型参数和训练数据规模不断增长, 智能模型训练对算力系统的规模和计算效率提出了更高的要求.

利用高性能计算系统进行并行智能训练, 是解决大规模智能模型训练问题, 提高智能模型训练计算效率和扩展性能的有效手段. 如图 1(a) 所示, 并行智能训练是人工智能、大数据、高性能计算的一个交叉领域. 经过多年的不断融合发展, 并行智能训练技术体系目前已初步形成. 如图 1(b) 所示, 并行智能训练在高性能并行智能训练体系结构基础上, 以并行智能训练框架为软件载体, 采用并行智能训练基本方法和关键技术提高智能训练系统的计算效率和可扩展性, 支持大规模人工智能模型的训练.

本文将围绕并行智能训练技术展开, 第 2 节主要介绍深度学习、智能处理器和高性能计算系统等背景与挑战; 第 3 节介绍数据并行、模型并行、混合并行等并行智能训练基本模式; 第 4 节介绍并行智能训练关键技术以及业界的最新研究进展; 第 5 节总结并行智能训练技术和框架软件发展面临的挑战问题; 第 6 节概述并行智能训练技术的发展趋势; 第 7 节介绍作者团队在并行智能训练框架方面的研究成果; 第 8 节为总结和展望.

2 背景与挑战

2.1 深度学习技术

以深度神经网络为代表的人工智能技术推动着计算机视觉 [1, 9, 10]、自动驾驶 [11, 12]、自然语言处理 (natural language processing, NLP) [13~18]、语音识别 [19, 20] 等智能应用的成功落地. 这些深度神经网络通过在海量数据上训练, 在许多任务上甚至达到超过人类的水平, 产生了一系列深度学习技术的应用实例, 例如目标检测与语义分割 [21]、图片分类 [10]、问答系统 [22]、图像生成 [23] 等.

然而, 近年来模型的参数量和训练数据量急剧增加, 大模型和大数据问题给深度学习系统的设计

2) Baidu allreduce. Github, 2022. https://github.com/baidu-research/baidu-allreduce.

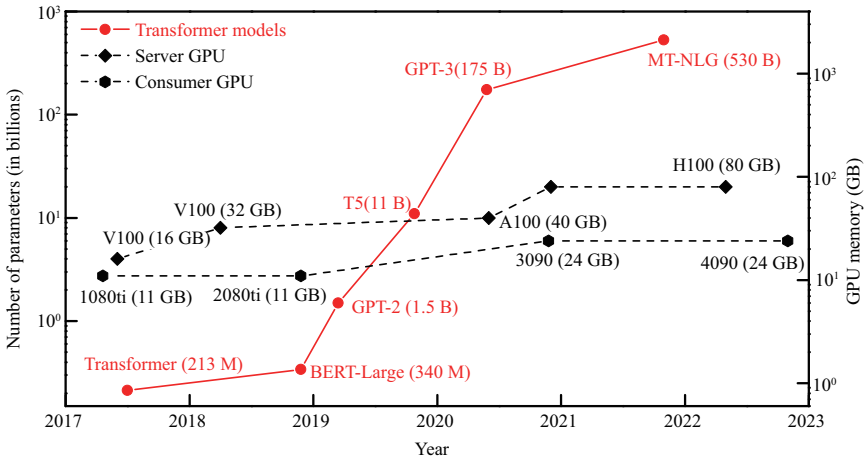

图 2 (网络版彩图) 大规模模型参数量 [3,6~8,24,25] 与 GPU (消费级和服务器级) 内存的增长趋势

Figure 2 (Color online) The growth trend of large-scale model [3,6~8,24,25] parameters and GPU memory

和实现带来了巨大挑战, 特别是基于 Transformer 的预训练模型 [3,6,7,24,25] 参数量已超过十亿 (billion) 数量级, 而且模型规模的增长速度非常快, 如图 2 [3,6~8,24,25] 所示, 近年发布的预训练模型参数量增长了超过 1000 倍.

此外, 训练的数据量往往达到 Terabyte (TB) 数量级. 例如图片数据集 JFT-3B 包含近 30 亿张图片 [26], 更大的 Common Crawl 网页数据集中的 token 数量达到了上千亿, 存储图像数据集 Open Image 需要 18 TB[3], 视频分析的 Youtube-8M 数据集甚至需要 1 Petabyte (PB) [27]. 典型的深度神经网络训练步骤需要迭代进行前向计算 (forward pass, FP) 和反向传播 (backward propagation, BP) 来获得更高的准确性, 而计算量与参数量和数据量呈正相关关系, 巨大的参数量和训练数据量会带来高昂的计算开销. 2012 年以来, 智能模型计算量每年增长 10 倍 [3,6,7,24,25], 增长速度超越摩尔定律. 因此亟须高性能并行智能训练系统支撑模型开发者在海量数据上更快地训练大规模神经网络模型 [28], 并行智能训练技术成为学术界和产业界的研究热点 [4,29,30].

2.2 智能训练处理器

智能模型训练过程主要包括神经网络的前向和反向计算, 其中涉及大量高维矩阵运算, 通用 CPU 难以应对这类计算密集型任务, 而 GPU, FPGA 等加速芯片并行计算的特性使其被广泛用于深度学习模型训练任务. 因此, 目前并行智能训练系统通常是异构的, 包含了 CPU 和专用硬件加速芯片. 这种异构计算系统由 CPU 访问主存或 I/O 设备读取数据, 将数据传输给硬件加速器进行模型训练. 以智能训练系统中广泛应用的英伟达 GPU 为例, 其配合使用 Compute Unified Device Architecture (CUDA) 套件进行矩阵加速计算, 其代表性产品 NVIDIA A100 的 Tensor Cores 可以在面向智能训练的数据格式 Tensor Float (TF32) 上进行计算, 提供高达 312 teraFLOPS (TFLOPS) 的计算能力. 与此同时, 谷歌公司也研发了应用于大规模神经网络计算的 TPU [31]. 国内也抓住 AI 芯片这一新的机遇进行投入, 如华为推出 Davinci 架构的 Ascend NPU 芯片、寒武纪研发 Cambricon-X [32]、百度发布昆仑系列芯片 [33] 等. 这些专用加速芯片极大提高了神经网络训练和推理的计算速度. 除此之外, 新一代高性能计算 (high-performance computing, HPC) 系统中的异构多核处理器, 例如国防科技大学的 MT-3000 [34],

3) Open Images V6. https://storage.googleapis.com/openimages/web/index.html.

表 1 高性能并行计算系统训练 ResNet50 模型的训练时间与模型精度 [35]

Table 1 Training time and model accuracy of the training ResNet50 model [35] in HPC systems

Time	Organization	Platform	System size	Training time	Accuracy (%)
2017–06	Facebook	NVIDIA P100	256	1 h	76.30
2018–07	Tencent	NVIDIA P100	2048	6.6 min	75.80
2018–12	Google	TPU v3	1024	2.2 min	76.30
2019–02	SenseTime	NVIDIA V100	512	7.3 min	75.30
2019–09	HUAWEI	Ascend 910 (FP16)	1024	59.8 s	75.90
2020–07	Google	TPU v3	4096	28.2 s	76.07
2021–12	NVIDIA	NVIDIA A100	4320	20.8 s	76.27
2022–06	Google	TPU v4	4096	11.5 s	76.03

在加速科学计算任务的同时, 也可用于智能训练和推理任务.

2.3 并行智能训练系统

基于上述计算加速设备, 采用异构计算模式的高性能并行计算系统越来越广泛地用于训练大规模神经网络模型. 例如在计算机视觉任务方面, 多个厂商使用 HPC 系统在 ImageNet-1K 数据集上并行训练 ResNet-50, 加速效果如表 1 [35] 所示. Meta (原 Facebook) 使用 256 个 GPU, 以 8192 的 mini-batch 大小训练 ResNet-50 [36], 华为使用自主并行智能训练框架 MindSpore 和数千个 Ascend 910 芯片达到了 1024 petaFLOPS (PFLOPS) 的峰值性能 (FP16). Summit 是 TOP500[4] 排名中首个实现 E 级峰值性能的 HPC 系统, 在从 3.5 TB 气候图像中分割极端天气模式时, 可扩展到使用 27360 个 NVIDIA V100 GPU, 其半精度 FP16 计算峰值性能达到 1.13E+18 FLOPS [37]. 在预训练语言模型方面, NVIDIA 使用 2048 个 A100 GPU 的 SuperPOD 训练 BERT 模型, 耗时 49 s. Google 使用 4096 个 TPU v3 芯片训练 BERT, 耗时仅 23 s. 为了研发 GPT-3 语言模型, OpenAI 使用了微软 Azure 计算集群中的约一万个 V100 GPU.

2.4 并行智能训练技术挑战

随着深度学习算法的快速演进以及模型参数量和训练数据量的指数级增长, 智能模型训练面临模型结构多样、内存开销巨大、参数通信密集等严峻挑战.

模型结构多样. 伴随着人工智能算法高速演进, 以深度神经网络为基础的智能应用, 其应用领域在不断拓展. 然而不同的任务种类体现出不同的计算特点, 例如应用于视频等应用的实时目标检测模型要求较高的吞吐量 [21], 以 Transformer 结构 [8] 为基础的预训练语言模型包含众多大规模矩阵乘法. 与此同时, 如 2.3 小节所述, 智能训练系统快速发展, 可利用的计算规模不断增大. 如何高效利用训练系统资源, 满足多种智能训练任务的不同需求, 对并行智能训练方法提出了挑战. 例如语言模型任务中, 其对模型精度增加的需求导致模型参数量高速增长 [38], 使得单个设备无法容纳完整模型, 以及研究者发现将稠密模型转为稀疏化激活可以进一步增加模型的表征能力 [39], 使得多个模型之间的计算内容不再相同. 本文将在第 3 节对并行智能训练基本模式进行介绍.

内存开销巨大. 如图 2 所示, 与大规模模型的参数量增长速度相比, 常用智能计算加速器的内存容量增长较为缓慢. 因此, 如何在保证训练效率的同时高效地利用内存空间, 成为大规模智能模型训

4) TOP500 list. https://www.top500.org/lists/top500/list/2019/11/.

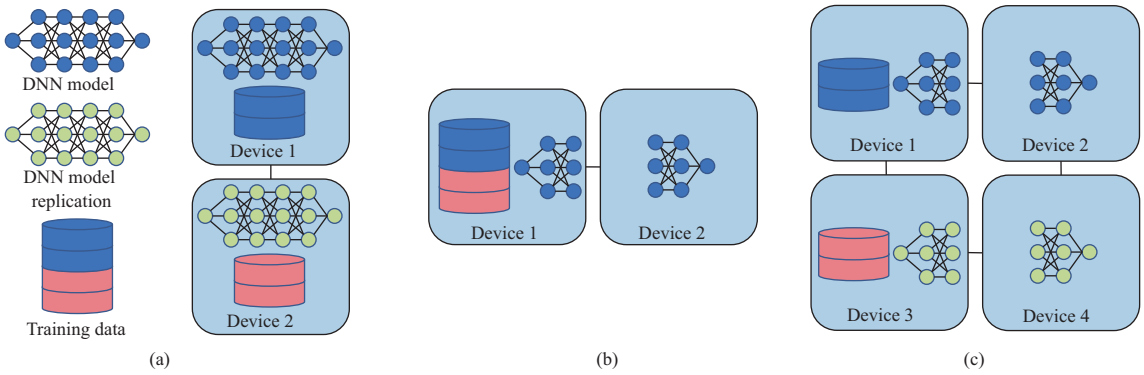

图 3 (网络版彩图) 并行智能训练基本模式示意图. (a) 数据并行; (b) 模型并行; (c) 混合并行

Figure 3 (Color online) Basic methods of parallel DNN training. (a) Data parallelism; (b) model parallelism; (c) hybrid parallelism

练的一大挑战. 首先, 单个设备的内存容量无法满足大规模模型训练的存储需求, 模型划分成为大规模模型训练的必需手段, 模型划分的优劣将显著影响智能训练运行效率, 然而, 随着模型并行技术的发展, 模型划分策略的搜索空间呈指数级增加, 给模型划分算法设计带来严峻的挑战. 其次, 常用异构计算系统有着多级存储空间且异构存储空间的传输带宽有着显著区别, 大容量低带宽存储空间的不当使用将降低整体训练性能, 本文第 4.1 和 4.2 小节将分别介绍模型划分和内存优化相关的关键技术.

参数通信密集. 随着异构加速设备的性能持续增长, 智能模型训练的通信频率不断提高, 同时智能模型参数量的增加也直接加大了参数通信的数据规模. 上述两方面的因素导致了参数通信开销在系统中占比增加, 参数通信开销成为了制约大规模并行智能训练效率提升的性能瓶颈. 一方面, 专门用于加速深度学习训练的专用芯片性能增长很快, 例如 NVIDIA A100 比其两代之前的产品 NVIDIA V100 的计算速度高 20 倍[5)]. 另一方面, 网络带宽增速较慢, 而参数通信数据量却越来越大. 高性能并行计算系统常用 Infiniband 等高速互连网络, 但其带宽提升相较于通信数据量提升较慢. 在并行智能训练系统的每次迭代中, 通信系统都需要传输模型参数或者梯度, 通信量与结点规模和模型大小呈正相关关系. 如第 2.1 和 2.3 小节所述, 在近年来结点规模和模型大小都增长的情况下, 通信量也增长明显. 智能模型训练中密集的参数通信严重影响了并行智能训练系统的吞吐率和可扩展性. 第 4.3 小节将介绍近期在参数通信优化方面的相关工作.

3 并行智能训练基本模式

随着智能模型参数量和训练数据量的增长, 单个计算设备已经不能满足模型训练的需求, 采用多个计算设备进行并行智能训练成为主流训练方式. 高效的并行智能训练, 要求在保证模型收敛性的前提下, 尽可能提高模型训练的吞吐量, 继而缩短模型的训练时间. 如图 3 所示, 按照数据划分和模型划分两个计算任务划分维度, 并行智能训练包括数据并行 [4,5] 和模型并行 [40~42] 两种基础并行训练模式, 以及混合多种并行模式的混合并行模式 [43~46]. 根据智能模型特点, 模型并行又演化出流水线并行 [40,43,47~50]、张量并行 [41,51~53]、专家并行 [42,54,55] 等多种计算模式.

为了描述不同并行智能训练模式的区别, 本文使用形式化方法描述并行智能训练中的数据划分、模型划分和参数通信等方式. 首先定义模型计算图为 $\mathcal{G}$, 训练数据集为 $\mathcal{S}$, 设备集合为 $\mathcal{D} = \{d_1, d_2, \ldots,$

5) NVIDIA A100 GPU. https://www.nvidia.com/en-us/data-center/a100/.

$d_{|\mathcal{D}|}\}$. 设备通信组集合为 $\mathcal{C}$. 定义模型计算图 $\mathcal{G}$ 到设备集合 $\mathcal{D}$ 的映射为 $f:\mathcal{G}\to\mathbb{P}(\mathcal{D})$, 数据集 $\mathcal{S}$ 到设备集合 $\mathcal{D}$ 的映射为 $g:\mathcal{S}\to\mathbb{P}(\mathcal{D})$, 其中 $\mathbb{P}(\mathcal{D})$ 表示设备集合 $\mathcal{D}$ 的幂集.

3.1 数据并行

数据并行 (data parallelism, DP) 训练是指将训练数据划分到多个训练单元, 多个训练单元之间按照一定规则定期同步模型参数实现并行训练的一种方式. 图 3(a) 展示了两个设备的数据并行训练方法. 形式上, 数据并行方法将模型计算图 $\mathcal{G}$ 中的算子 $O,\forall\, O\in\mathcal{G}$ 放置到设备集合 $\mathcal{D}$ 上. 对于数据集 $\mathcal{S}$, 数据并行方法将数据集 $\mathcal{S}$ 切分为 $|\mathcal{D}|$ 份子数据集 $\{S_1,S_2,\ldots,S_{|\mathcal{D}|}\}$, 即 $\mathcal{S}=\{S_1,S_2,\ldots,S_{|\mathcal{D}|}\}$, 然后将切分后的数据集 $\mathcal{S}$ 放置到设备集合 $\mathcal{D}$ 上, 且不同的子数据集 S_i 放置在不同的设备上. 数据并行方法的通信组集合 $\mathcal{C}$ 有且仅有一个通信组 $\mathcal{D}$, 即有 $\mathcal{C}=\{\mathcal{D}\}$, 在训练过程中, 通信组内的设备间通信方法一般为 AllReduce. 综上, 数据并行方法的模型计算图 $\mathcal{G}$ 与设备集合 $\mathcal{D}$ 的映射关系为

$$f(O)=\mathcal{D},\ \forall\, O\in\mathcal{G}. \tag{1}$$

数据集 $\mathcal{S}$ 与设备集合 $\mathcal{D}$ 的映射关系为

$$\begin{cases} g(S_i)\neq g(S_j),\ \text{if } i\neq j,\\ g(\mathcal{S})=\mathcal{D}. \end{cases} \tag{2}$$

DP 是最早被提出, 也是目前应用最广泛的并行智能训练方法. 早期的深度学习模型能够放置在单个设备上进行训练, 但是随着训练模型所需数据量的增加, 训练模型所需的时间也在不断增加. DP 将模型复制多份, 每一份模型参数放置在一个计算设备 (或多个计算设备组成的一个计算单元) 上. 在训练过程中, 将同时输入多个小批次 (mini-batch) 数据, 并让每个设备负责一个 mini-batch 的计算, 从而利用大量计算设备提高模型训练吞吐量, 减少模型训练时间. 由于不同设备上的模型处理着不同 mini-batch 的数据, 为了保证使用 DP 进行模型训练的收敛性, 常用的方法为在所有设备对当前批次的前向计算和反向计算完成后, 对所有设备上的梯度进行聚合, 使用聚合后的梯度去更新各个设备上的模型参数.

具体实现上, DP 有中心化和去中心化两种参数通信架构. 参数服务器 [5] (parameter server, PS) 是 DP 中心化实现方式中最常见的架构. 在 PS 架构下, 设备所在的结点被分为主结点 (server) 和工作结点 (worker), server 负责管理模型参数, 通过聚合从各个 worker 收集到的梯度来更新模型参数, worker 则从 server 处获取所需的模型参数, 并利用得到的模型参数对输入的批次数据进行计算, 得到对应的梯度. 相比于中心化实现, 去中心化实现中各个结点地位平等, 所有设备在反向计算后对梯度进行规约并更新模型参数, 不存在通信热点问题. Ring AllReduce 算法是 DP 去中心化实现的代表性方法, 算法将参与模型训练的设备组织成环, 在算法运行的每一步, 每个设备都会从左邻居收到数据块, 并向右邻居发送数据块, 保证了结点间带宽的充分利用. Ring AllReduce 算法将设备组织成环状结构, 当底层网络拓扑不是环状结构时, 效果不是很好, 因此一些工作 [56, 57] 针对不同的网络拓扑提出了特定的去中心化算法.

3.2 模型并行

模型并行 (model parallelism, MP) 训练是指将智能模型划分到多个计算单元的一种训练方式. 图 3(b) 展示了两个设备的模型并行训练方法. 目前常用的 MP 方法有流水线并行、张量并行以及专家并行等方法. MP 能够降低对单个设备的存储需求, 解决大模型无法在单个设备上训练的问题. 通

常来说, 模型参数量与模型容量成正比, 而模型容量一定程度上决定了模型在实际任务上的性能表现. 为了在实际任务上取得更好的性能, 深度学习模型参数量呈现非线性增长. 而随着模型参数量的增加, 单个设备已经无法满足模型训练时的存储需求, MP 成为满足大型模型存储需求的有效方法.

流水线并行 (pipeline parallelism, PP) 基于算子间 (inter-operator) 切分组织训练过程, 是目前最常用的并行智能训练方法之一. 形式上, 流水线并行方法将模型计算图 $\mathcal{G}$ 划分为 $|\mathcal{D}|$ 个连续的算子序列 $\{O_1, O_2, \ldots, O_{|\mathcal{D}|}\}$, 即 $\mathcal{G} = \{O_1, O_2, \ldots, O_{|\mathcal{D}|}\}$, 其中 $O_i, 1 \leqslant i \leqslant |\mathcal{D}|$ 包含模型计算图 $\mathcal{G}$ 拓扑排序结果中一部分连续的算子, 且满足 $O_i \bigcap O_j = \emptyset$, if $i \neq j$, 然后将各个算子序列放置到设备集合 $\mathcal{D}$ 上, 且不同的算子序列 O_i 放置到不同的设备上. 对于数据集 $\mathcal{S}$, 流水线并行方法将数据集 $\mathcal{S}$ 中的数据 $s, \forall\ s \in \mathcal{S}$ 输入到模型计算图 $\mathcal{G}$ 切分后的第一个算子序列 O_1 所放置的设备上. 流水线并行方法的通信组集合 $\mathcal{C}$ 有且仅有一个通信组 $\mathcal{D}$, 即有 $\mathcal{C} = \{\mathcal{D}\}$, 在训练过程中, 各个通信组内的设备间通信方法一般为点对点 (peer to peer, P2P) 通信. 综上, 流水线并行方法的模型计算图 $\mathcal{G}$ 与设备集合 $\mathcal{D}$ 的映射关系为

$$\begin{cases} f(O_i) \neq f(O_j), \text{ if } i \neq j, \\ f(\mathcal{G}) = \mathcal{D}. \end{cases} \tag{3}$$

数据集 $\mathcal{S}$ 与设备集合 $\mathcal{D}$ 的映射关系为

$$g(s) = f(O_1),\ \forall\ s \in \mathcal{S}. \tag{4}$$

张量并行 (tensor parallelism, TP) 基于算子内 (intra-operator) 切分组织训练过程, 是模型单层所需存储空间大于设备存储空间时训练模型的有效解决方案. 对于模型中的算子, TP 结合算子的特性以及训练模型所用设备的特性进行切分, 将算子分配给多个设备进行处理, 降低设备的计算存储负载. 形式上, 张量并行方法将模型计算图 $\mathcal{G}$ 的算子 $O, \forall\ O \in \mathcal{G}$ 切分为 $|\mathcal{D}|$ 个子算子 $\{o_1, o_2, \ldots, o_{|\mathcal{D}|}\}$, 即 $O = \{o_1, o_2, \ldots, o_{|\mathcal{D}|}\}$, 然后将切分后的算子 O 放置在设备集合 $\mathcal{D}$ 上, 且不同的子算子 o_i 放置在不同的设备上. 此外, 张量并行将数据集 $\mathcal{S}$ 中的数据 $s, \forall\ s \in \mathcal{S}$ 输入到设备集合 $\mathcal{D}$. 张量并行方法的通信组集合 $\mathcal{C}$ 有且仅有一个通信组 $\mathcal{D}$, 即有 $\mathcal{C} = \{\mathcal{D}\}$, 在训练过程中, 各个通信组内的设备间通信方法根据算子的切分方式可能为 AllReduce 或者 AllGather + ReduceScatter. 综上, 张量并行方法的模型计算图 $\mathcal{G}$ 与设备集合 $\mathcal{D}$ 的映射关系为

$$\begin{cases} f(o_i) \neq f(o_j), \text{ if } i \neq j, \\ f(O) = \mathcal{D},\ \forall\ O \in \mathcal{G}. \end{cases} \tag{5}$$

数据集 $\mathcal{S}$ 与设备集合 $\mathcal{D}$ 的映射关系为

$$g(s) = \mathcal{D},\ \forall\ s \in \mathcal{S}. \tag{6}$$

专家并行 (expert parallelism, EP) 是针对混合专家 (mixture-of-experts, MoE) 模型结构特性设计的并行智能训练方法. 近年来, 虽然以 GPT-3 [3] 为代表的大规模深度学习模型在众多实际任务上取得了优异的性能表现, 但是训练这些大模型需要花费大量的时间. 为了在合理的时间内训练出性能优异的大模型, 近期的工作 [55,58] 提出了 MoE 模型结构. MoE 模型由专家层和非专家层构成 [39,58], 目前常见的 MoE 模型是通过将 Transformer 结构中的全连接层替换成专家层, 将自注意力层以及门控网络组成非专家层产生的. 专家层中的专家通常由简单的前馈神经网络组成, 对于每个输入样本, MoE 模型通过门控网络为其选择固定个数的专家 (本文中以选择一个专家为例). 因此, 随着专家层专家个

数的增多, 模型总体参数量将不断增大, 但计算量几乎不变, 控制了模型的计算开销, 实现了模型参数量与计算量的解耦, 达到短时间内训练大模型的目的.

用 $\mathcal{E}$ 表示 MoE 模型中的专家集合, 一个 MoE 模型包含 $|\mathcal{D}|$ 个专家 $\{e_1, e_2, \ldots, e_{|\mathcal{D}|}\}$, 即有 $\mathcal{E} = \{e_1, e_2, \ldots, e_{|\mathcal{D}|}\}$, 专家并行方法将专家集合 $\mathcal{E}$ 放置到设备集合 $\mathcal{D}$ 上, 且不同的专家 e_i 放置到不同的设备上. 对于数据集 $\mathcal{S}$, 专家并行会利用 Gate 函数将其划分为 $|\mathcal{D}|$ 个子数据集 $\{S_1, S_2, \ldots, S_{|\mathcal{D}|}\}$, 即 $\mathcal{S} = \{S_1, S_2, \ldots, S_{|\mathcal{D}|}\}$, 然后将切分后的数据集 $\mathcal{S}$ 放置到设备集合 $\mathcal{D}$ 上, 且不同的子数据集 S_i 放置在不同的设备上. 专家并行方法的通信组集合 $\mathcal{C}$ 有且仅有一个通信组 $\mathcal{D}$, 即有 $\mathcal{C} = \{\mathcal{D}\}$, 在训练过程中, 各个通信组内的设备间通信方法为 AlltoAll. 综上, 专家并行方法中专家集合 $\mathcal{E}$ 与设备集合 $\mathcal{D}$ 的映射关系可以表示为

$$\begin{cases} f(e_i) \neq f(e_j), \text{ if } i \neq j, \\ f(\mathcal{E}) = \mathcal{D}. \end{cases} \tag{7}$$

数据集 $\mathcal{S}$ 与设备集合 $\mathcal{D}$ 的映射关系为

$$\begin{cases} g(S_i) \neq g(S_j), \text{ if } i \neq j, \\ g(\mathcal{S}) = \mathcal{D}. \end{cases} \tag{8}$$

3.3 混合并行

混合并行 (hybird parallelism) 是指结合模型结构特点或智能计算系统体系结构特点, 利用多种并行训练方法进行大规模深度学习模型的多维度并行训练. 图 3(c) 展示了 4 个设备上混合数据并行与模型并行的训练方法. 前文提及的各种并行智能训练方法有其独特优势以及使用场景, 通过结合这些方法, 混合并行方法能够实现高效的模型训练. 例如, 在利用高带宽的计算结点进行模型训练时, 结点内使用 TP 将模型的计算存储负载均匀分配给结点内的设备, 而在结点间使用 PP 对模型负载进行划分, 最后再利用 DP 来增大模型的训练规模, 这种方式下, 可以充分利用结点内的高带宽来应对 TP 中较大的集合通信开销, 而使用结点间相对较低的带宽应对 PP 中相对较小的流水级之间的通信开销, 并使用 DP 来充分利用集群中大量的计算结点, 最终达到高效模型训练的目标.

目前广泛使用的混合并行方式包括混合 DP 与 PP 的二维混合并行训练 [48, 49, 59], 以及混合 DP, TP 以及 PP 的三维混合并行 [43, 60]. 此外, EP 结合 DP, EP 结合 TP 等混合并行方式也常被应用到 MoE 类大模型的训练. 以目前 GPT-3 等超大模型并行训练中常见的数据并行、张量并行结合流水线并行的混合并行方法为例, 假设流水线级数为 p, 张量并行维度为 t, 数据并行维度为 q, 且满足 $p \times t \times q = |\mathcal{D}|$. 首先将模型计算图 $\mathcal{G}$ 分为 p 个连续的不相交算子序列 $\{O_1, O_2, \ldots, O_p\}$, 即 $\mathcal{G} = \{O_1, O_2, \ldots, O_p\}$. 对于 $O_j, 1 \leqslant j \leqslant p$ 中的每个算子 $O, \forall\, O \in O_j$, 与张量并行中类似, 我们将其切分为 t 个子算子, 即 $O = \{o_1, o_2, \ldots, o_t\}$.

对于设备集合 $\mathcal{D}$, 我们将其组织成三维的形式, 即有 $\mathcal{D} = \{d_{ijk} | 1 \leqslant i \leqslant q, 1 \leqslant j \leqslant p, 1 \leqslant k \leqslant t\}$. 在此基础上, 我们在设备集合 $\mathcal{D}$ 上定义数据并行通信组集合 A, 流水线并行通信组集合 B, 以及张量并行通信组集合 C, 每个通信组由一些设备构成, 因此也可以视为设备集合. 对于数据并行通信组集合 A, 我们首先定义其所包含的通信组 $A_{jk} = \{d_{ijk} \in \mathcal{D} | 1 \leqslant i \leqslant q\}$, 然后定义数据并行通信组集合 $A = \{A_{jk} | 1 \leqslant j \leqslant p, 1 \leqslant k \leqslant t\}$. 对于流水线并行通信组集合 B, 我们首先定义其所包含的通信组 $B_{ik} = \{d_{ijk} \in \mathcal{D} | 1 \leqslant j \leqslant p\}$, 然后定义流水线并行通信组集合 $B = \{B_{ik} | 1 \leqslant i \leqslant q, 1 \leqslant k \leqslant t\}$. 对于张量并行通信组集合 C, 我们首先定义其所包含的通信组 $C_{ij} = \{d_{ijk} \in \mathcal{D} | 1 \leqslant k \leqslant t\}$, 然后定义张量并行通信组集合 $C = \{C_{ij} | 1 \leqslant i \leqslant q, 1 \leqslant j \leqslant p\}$.

在上述基础上, 混合并行方法将模型计算图 $\mathcal{G}=\{O_1,O_2,\ldots,O_p\}$ 放置到数据并行通信组集合 A 的元素的并集构成的集合 $\{\bigcup_{1\leqslant k\leqslant t}A_{1k},\bigcup_{1\leqslant k\leqslant t}A_{2k},\ldots,\bigcup_{1\leqslant k\leqslant t}A_{pk}\}$ 上, 且不同的 O_j 放置到上述集合的不同元素上. 对于算子序列 $O_j,1\leqslant j\leqslant p$, 其放置到的设备集合为 $f(O_j)$, 且有 $|f(O_j)|=q\times t$. 我们令 $f(O_j)=\{A_{l1},A_{l2},\ldots,A_{lt}\}$, 其中 l 根据映射关系为闭区间 $[1,p]$ 中的某个整数. 对于 O_j 中的算子 $O=\{o_1,o_2,\ldots,o_t\},\forall\, O\in O_j$, 我们将算子 O 放置到 $f(O_j)$ 所包含的设备上, 且不同的子算子 o_k 放置在 $f(O_j)$ 的不同元素上. 数据集 $\mathcal{S}$ 将被切分为 q 个子数据集 $\{S_1,S_2,\ldots,S_q\}$, 即 $\mathcal{S}=\{S_1,S_2,\ldots,S_q\}$, 根据设备集合 $\mathcal{D}$ 的组织方式, 我们可以令 $f(O_1)=\{C_{1r},C_{2r},\ldots,C_{qr}\}$, 其中 r 根据映射关系为闭区间 $[1,p]$ 中的某个整数. 训练过程中, 混合并行方法会将切分后的数据集 $\mathcal{S}$ 输入到模型计算图 $\mathcal{G}$ 切分后的第一个算子序列 O_1 所放置的设备集合 $f(O_1)$ 上, 且不同的子数据集 S_i 放置在 $f(O_1)$ 的不同元素上. 对于子数据集 $S_i,1\leqslant i\leqslant q$ 中的数据 s, 混合并行方法将其输入到子数据集 S_i 输入的设备集合 $g(S_i)$ 上. 混合并行方法的通信组集合 $\mathcal{C}=A\bigcup B\bigcup C$, 即混合并行方法的通信组集合是数据并行通信组集合 A, 流水线并行通信组集合 B, 以及张量并行通信组集合 C 的并集.

综上, 混合并行方法的模型计算图 $\mathcal{G}$ 与设备集合 $\mathcal{D}$ 的映射关系为

$$\begin{cases} f(O_i)\neq f(O_j),\ \text{if}\ i\neq j,\\ f(\mathcal{G})=\left\{\displaystyle\bigcup_{1\leqslant k\leqslant t}A_{1k},\ \bigcup_{1\leqslant k\leqslant t}A_{2k},\ldots,\ \bigcup_{1\leqslant k\leqslant t}A_{pk}\right\},\\ f(o_i)\neq f(o_j),\ \text{if}\ i\neq j,\\ f(O)=f(O_j),\ \forall\, O\in O_j,1\leqslant j\leqslant p. \end{cases} \tag{9}$$

数据集 $\mathcal{S}$ 与设备集合 $\mathcal{D}$ 的映射关系为

$$\begin{cases} g(S_i)\neq g(S_j),\ \text{if}\ i\neq j,\\ g(\mathcal{S})=f(O_1),\\ g(s)=g(S_i),\ \forall\, s\in S_i,1\leqslant i\leqslant q. \end{cases} \tag{10}$$

4 并行智能训练关键技术

并行智能训练基本模式为智能模型并行训练提供了基本方法, 但是, 如何根据智能模型和训练系统的特点, 对并行智能训练方法进行优化, 尽可能挖掘并行训练系统的潜力, 提高并行智能训练的计算效率和扩展性能, 仍然面临模型划分、内存优化及通信优化等并行智能训练关键技术问题.

4.1 模型划分技术

在模型并行方法中, 多个计算设备将共同训练同一个模型, 模型划分方式对目标模型的计算效率和通信效率等方面有显著影响, 图 4 所示的模型划分技术是一项提高并行训练效率的关键技术, 其将模型结构和设备拓扑作为输入, 使用特定的算法获得可分布式运行的模型子图. 深度学习模型训练的计算图可以视为一个有向无环图 (directed acyclic graph, DAG), 文献 [61,62] 指出, 找到有着最佳训练效率的有向无环图划分是一个 NP-hard 问题, 为了求得这一问题的近似解, 研究者们在不同的并行训练场景下, 提出了不同的模型划分策略搜索算法.

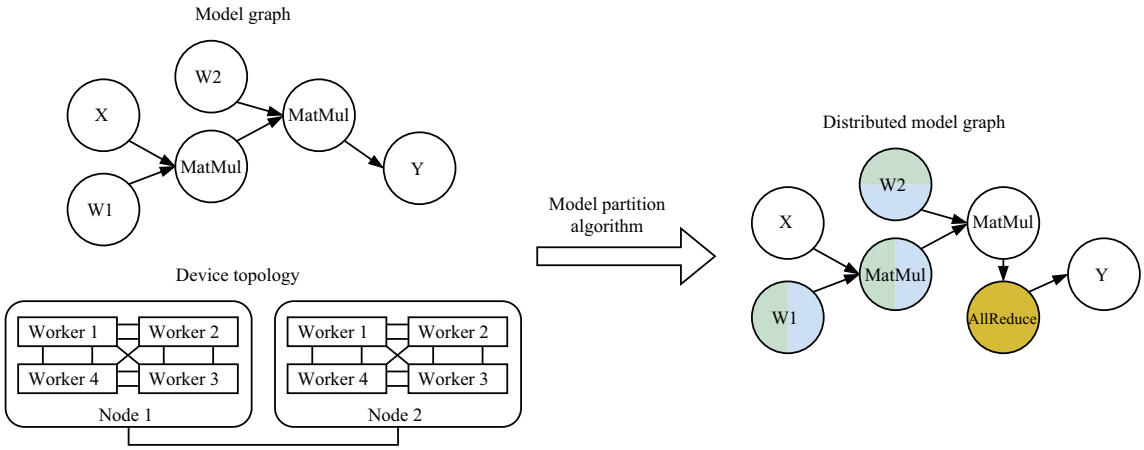

图 4 (网络版彩图) 模型划分技术示意图. 模型计算图中的箭头代表了数据之间的依赖关系, 设备拓扑中的连线代表了设备间的通路

Figure 4 (Color online) An overview of model partitioning. Arrows in the model graph represent the data dependencies. Connections in the device topology represent the paths between devices

4.1.1 算子间模型划分

在流水线并行中, 模型划分技术的主要目标是平衡各流水级的计算负载以及降低流水级间的通信量, 从而提高设备利用率以及通信效率. 针对数据并行与流水线并行结合的混合并行方法, DAPPLE [49] 设计了 DAPPLE Planner, 其结合设备间拓扑以及模型结构, 使用动态规划算法求解最优的模型划分策略. 针对常见的数据并行、流水线并行结合张量并行的三维混合并行方法, Piper [63] 提出了将单个样本处理时间 (time-per-sample, TPS) 最小化的优化目标, 为解出最优的模型划分策略, Piper 首先设计了一个启发式算法求解出各种情况下的局部 TPS 最小值, 再结合动态规划算法求解全局 TPS 最小值, 最后通过回溯构造出全局最优解对应的模型划分策略. AutoPipe [61] 发现在应用流水线并行方法时, 流水线的关键路径 (critical path) 对流水线的吞吐量有着一定的影响, 并且使用关键路径可以大大减小流水线并行划分策略的搜索空间, 因此 AutoPipe 提出了利用关键路径求解最优模型划分策略的启发式算法. 除了上述自动求解模型划分策略的划分技术外, FTPipe [64] 根据深度学习模型前反向计算以及内存优化技术的相关特点, 提出了应用流水线并行时的固定模型划分策略, 并通过实验验证了这种固定划分策略的加速效果. HPH [59] 则针对异构计算环境设计了一个线性规划算法, 在考虑异构设备间不同计算能力的情况下平衡各流水级的负载. 上述针对流水线并行的模型划分技术中, 模型划分在算子间 (inter-operator) 或者多个连续算子组成的模型层 (如 Transformer 层 [8] 和卷积残差模组 [1]) 间进行, 模型的数据并行及张量并行形式保持一致.

4.1.2 算子内模型划分

在张量并行中, 模型划分技术将在算子内 (intra-operator) 进行, 以最大化训练效率为目标进行模型划分. OptCNN [65] 首先考虑算子输入所有可能的划分维度, 以计算机视觉任务的数据输入为例, OptCNN 会考虑模型各算子输入数据的批次数量、宽度、高度和通道数, 数据沿着这些维度可以被划分至不同的计算设备上, 随后 OptCNN 建立了一个代价模型来估算任意一种划分策略的运行效率, 最后使用动态规划算法得到最佳的模型划分策略, 但是 OptCNN 只能处理线性结构的模型. TensorOpt [66] 将动态规划算法拓展至非线性的模型, 然而巨大的搜索空间会导致算法运行时间过长, 因此 TensorOpt

将算子进行分组并采用前沿跟踪线性动态规划 (frontier tracking linear dynamic programming) 算法来降低并行策略搜索的复杂度. 除此之外, FlexFlow [67] 在 OptCNN 的基础上进一步探索了神经网络在更多维度上的模型划分, 并使用随机马尔可夫链蒙特卡洛 (Markov chain Monte Carlo, MCMC) 算法为神经网络中的每个算子寻找最优的并行策略.

在同时应用算子内与算子间模型划分技术时, 模型划分方法的搜索空间将进一步扩大, 给算法设计带来更大的挑战. Alpa [68] 发现可以使用分层的方式搜索算子内和算子间的模型划分策略, 并将分层的搜索空间映射到计算集群的分层结构中. 首先, Alpa 采用了用于描述计算集群拓扑的设备网格 (device mesh). 随后, Alpa 设计了一个双层嵌套的搜索算法, 内层为一个接收设备网格并搜索最优的算子内模型划分方案的整数线性规划算法, 外层则使用一个以整数线性规划算法的结果为优化项的动态规划算法, 找到将设备划分至设备网格以及模型划分至各设备网格的最优解, 即完成算子间的模型划分. Unity [69] 则认为模型划分均可以使用子图替换完成. 具体而言, Unity 根据基本并行方法以及计算优化方法定义了一系列的计算子图替换规则, 然后使用一个三级分层的搜索算法, 在计算图分割、子图替换以及子图设备分配 3 个方面进行策略的搜索, 最终获得一个较优的计算图划分策略.

4.2 内存优化技术

模型规模的高速增长加大了模型训练的高速内存需求, 内存溢出 (out of memory, OOM) 是大型模型训练中经常出现的错误. 智能训练系统中设备高速内存的总量, 成为制约智能训练系统对大型模型支撑能力的主要因素. 如何对智能训练过程中的内存使用进行调度优化, 提高高速内存的使用效率以及智能训练系统对大型模型的支撑能力, 成为并行智能训练技术的研究热点.

4.2.1 单进程内存优化

显存交换 (swap) 是一种利用 CPU 内存来保存中间计算结果以节省 GPU 内存空间的方法, 如图 5(a) 所示, 其主要思想是在前向传播中将计算得到的激活值从 GPU 内存卸载到 CPU 内存中, 在反向传播的时候再将所需要的激活值加载到 GPU 内存中. 得益于 GPU 计算任务与传输任务的可并行性, swap 方法有不会占用 GPU 额外的计算资源的优点, 但受限于 PCIe 链路的带宽, 卸载与加载的时间开销往往较大. swap 方法被 vDNN [70] 首先应用在计算机视觉领域的模型训练中, 虽然 vDNN 能够降低 GPU 内存的使用量, 但是其并没有很好地考虑不同卷积层的特征, 只是粗略地对卷积层使用 swap 策略. moDNN [71] 对该方面进行了改进, 其根据不同卷积函数所需内存空间的特性选择不同的 swap 策略. SwapAdvisor [72] 利用遗传算法对模型的训练过程进行建模, 从而自动化地求得最优的 swap 策略. 然而, 随着自然语言处理领域模型的发展, 自注意力层得到了广泛应用, 自注意力层相较于卷积层而言拥有更大激活值和更低计算时间, 这一特性使得 swap 方法的应用开销变得很大.

重计算 (recomputation) 是一种以时间换空间的优化方法, 如图 5(b) 所示, 其主要思想是利用前向传播计算的中间结果 (即模型激活值) 在反向传播时才会被使用这一特点, 在前向过程中将部分激活值直接丢弃, 在反向传播需要时再重新进行前向传播的计算以获得所需的激活值. 重计算的优点是能够立刻释放大量内存, 但是其缺点同样也很突出, 因为其在进行反向传播时需要再进行一次前向计算, 这将产生额外的计算开销. 重计算技术发表于 Checkpoint [73] 中, 该方法需要手动设置需要保存的中间激活值, 不同的激活值丢弃策略会导致重计算时不同的计算开销. Checkmate [74] 通过对模型的训练过程进行建模, 利用整数线性规划方法来求解需要保存的中间激活值, 对于静态模型可以求得较优解. DTR [75] 则不需要提前对模型训练进行策略求解, 而是在训练运行时通过启发式函数来动态选择需要丢弃的激活值.

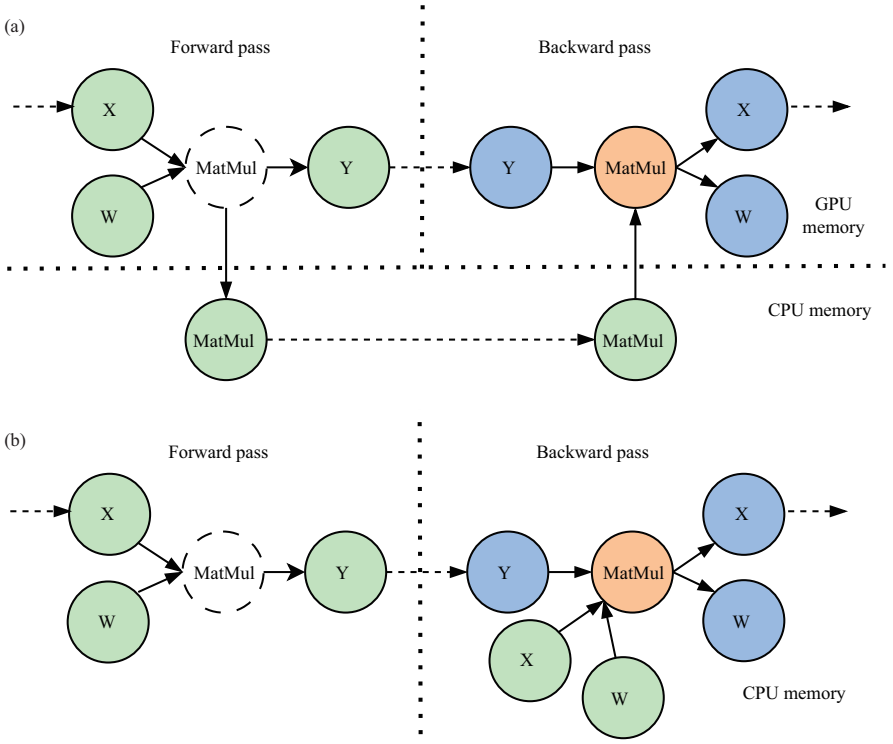

图 5 (网络版彩图) (a) 显存交换和 (b) 重计算技术示意图. 两个技术可以降低智能模型训练过程中矩阵计算激活值的存储开销

Figure 5 (Color online) Illustration of (a) swap and (b) recomputation. Both methods reduce the memory overhead of activation of the matmul operator

SuperNeurons [76] 首次将重计算和 swap 方法结合到一起, 其考虑到卷积层计算时间长但占用空间小, 而池化层、归一化层等层计算时间短但占用空间大的特点, 对卷积层采取 swap 方法而对池化层等层采取重计算方法来释放 GPU 内存. Capuchin [77] 则进一步对 SuperNeurons 的方法进行改进, 通过对模型的训练过程进行分析, 比较不同层的内存优化开销, 来确定最终的内存优化策略. DELTA [78] 则进一步利用启发式函数来动态地确定训练时的内存优化策略. 此外, MPress [79] 在结合重计算和 swap 方法的基础上, 提出了利用其他 GPU 空闲内存加速训练的方法, 为运行时内存优化提供了新思路.

4.2.2 多进程内存优化

在多计算设备参与的多进程并行智能训练中, 内存优化技术有了新的空间. 在数据并行训练中, 零冗余优化器 (ZeRO) [80] 内存优化技术将模型的优化器状态、梯度和参数划分至各个工作设备上, 从而消除冗余的存储开销. ZeRO-Offload [81] 在 ZeRO 的基础上, 将在 GPU 上保存的优化器状态和梯度的切分卸载至内存, 并且在整个训练阶段将优化器状态维持在内存中. ZeRO-Infinity [82] 则进一步将 NVMe 存储加入到数据并行训练系统之中, 使模型参数可扩展到更大规模. 在使用张量并行训练 Transformer 模型时, 有部分算子未被划分, 序列并行技术 [83] 认为这些算子的计算是冗余的, 通过把 AllReduce 通信替换成 ReduceScatter-AllGather 通信对, 其将完整算子的输入沿着序列维度进行划分, 降低了这部分算子的存储与计算开销.

4.3 参数通信优化技术

采用多个计算设备对智能模型进行分布式并行智能训练, 需要在计算设备间进行大量的模型参数通信. 此外, 模型参数规模的增长以及计算设备性能的提高, 使得参数通信量和参数通信频率也进一步增加. 参数通信优化技术成为提高大规模并行智能训练扩展性能的主要技术.

4.3.1 通信频次压缩技术

基于数据并行的随机梯度下降算法 (data parallel SGD, D-SGD) [4,5] 需要在训练进程之间频繁地进行参数通信, 以同步各个训练进程中的模型. 在 D-SGD 的每次迭代中, 计算结点 $i \in \{1,\ldots,W\}$ 采样 mini-batch 数据 $X_t^i \subset \{x_1,\ldots,x_n\}$, 通过反向传播算法计算局部梯度 $g_t^i = \frac{1}{|X_t^i|}\sum_{x_j \in X_t^i} \nabla f_i(\boldsymbol{\omega}_t, x_j)$, 随后, 各个计算结点或聚合梯度, 或同步更新后的参数, 其中梯度通信最为常见 (本文把并行智能训练中梯度和参数的通信统称为参数通信). 这种参数通信是深度学习模型分布式并行智能训练中不可避免的开销. 我们将参数通信总数据量定义为 $T \times G$, 其中 T 表示迭代次数, G 表示每次通信的数据量. 由于 D-SGD 训练大型模型时的迭代次数较多, 模型的梯度数据量 (参数量) 较大, 因此系统的通信量很大, 系统中通信 – 计算比将决定并行智能训练系统的加速比以及可扩展性.

在单次通信量不变的前提下减少通信频次 T 可以有效地减少通信总量. 一般的 D-SGD 算法在每次模型计算出梯度后都需要进行参数通信, 而减少通信频次之后, 模型经过多次更新才进行一次参数同步通信, 这样可以缩小总体通信开销 $T \times G$, 进而提升系统效率. McDonald 等 [84] 提出了基于模型平均的随机梯度下降算法 (MA-SGD). MA-SGD 中所有结点执行多次迭代, 在本地更新模型参数, 然后将其发送到中心结点, 这样经过若干次迭代才进行一次参数同步通信. 基于异步更新的随机梯度下降算法 (asynchronous stochastic gradient descent, ASGD) [85~87] 允许各个计算结点独立地向中心结点推送梯度、拉取模型参数, 这样一些较快的计算结点不需要等待较慢的结点, 即可进行通信. 考虑到 ASGD 可能减慢全局模型的收敛速度, 一些研究人员提出延时同步并行训练算法 (stale synchronous parallel, SSP) [88,89]. 这类训练算法允许一定计算结点之间有一定的的通信间隔, 以改善模型收敛速度.

4.3.2 梯度压缩技术

通过 mini-batch 数据计算出的梯度数值是整个数据集上梯度的无偏估计, 其本身具备随机性, 且数值接近零, 具备较大的压缩潜力. 一些研究者提出基于梯度压缩的 D-SGD 算法, 这些梯度压缩方法可以分为三种类型: 量化、稀疏化和低秩近似.

梯度量化. 一种简单的梯度压缩思路是 1 bit 量化 (1-bit quantization) [90], 这种方法每次只传输原始梯度的符号位信息, 由于符号位可以用 1 bit 压缩, 相比 32 bit 的浮点数值表示压缩了 32 倍. 但是这种量化思路无法保持原始数值的数学期望, 往往带来模型训练的精度损失. 为解决此问题, QSGD [91] 提出广泛使用的 Stochastic Rounding 技术, 使得随机量化产生无偏估计, 并给出了模型训练的收敛速度证明 [91~94]. TernGrad [92], SignSGD [93] 和 3LC [95] 等工作则在 QSGD 的基础上, 将梯度进行分类量化. 除此之外, SketchML [96] 使用 Quantile Sketch 对梯度进行量化. 一些梯度量化工作研究如何在保证模型训练精度的前提下, 提高量化方法的压缩率. 例如, TINYSCRIPT [97], ALQ 和 AMQ 算法 [98] 通过计算梯度分布的最佳量化级别来最小化量化的压缩误差. Bai 等 [99] 则设计了压缩 – 通信的成本模型, 通过任务调度来优化并行智能训练系统中压缩梯度的计算开销.

梯度稀疏化. 模型训练过程中, 梯度数值不仅绝大部分接近零值, 且分布集中. 这表明大量的模型参数在一次迭代中不需要都进行更新. 利用模型训练的这一特点, Ström [100] 提出了梯度稀疏化方法, 使用一定阈值筛选部分梯度进行通信, 并在部分模型上实现了较高的压缩率. TopK 方法 [101~106] 是常

用的梯度稀疏化方法, 这类压缩方法只筛选那些具有显著的梯度值的梯度进行通信. 梯度误差补偿技术 (Error-Feedback) 是降低模型稀疏化训练算法精度损失的关键, 其思路是将稀疏化误差累积到后续迭代的梯度中. Karimireddy 等 [94] 在理论上证明了 Error-Feedback 算法的收敛速度. MIPD [107] 将稀疏化梯度进一步量化, 为改善高压缩率导致的模型收敛延缓问题, 其在压缩梯度时考虑了模型参数的重要性.

低秩近似. 一些研究基于梯度矩阵的信息冗余, 提出将梯度矩阵分解为通信量较小的矩阵 [108~110] 的低秩近似方法. 梯度矩阵低秩近似方法中, PowerSGD 实现了高达 128 倍的压缩率. 但是梯度压缩算法的压缩率过高可能导致模型精度降低, 而较低的压缩率又无法满足进一步减少通信量的需求. 自适应的梯度压缩策略 ACCORDION [111] 针对稀疏化与低秩近似方法, 利用模型训练的不同时期对压缩率的不同需求, 通过检测关键学习区 (critical learning regimes) 来自动调整模型压缩率.

4.3.3 集合通信优化技术

拓扑感知的集合通信. 并行智能训练过程中需要大量的集合通信操作. 面向并行智能训练系统的通信库, 如 NVIDIA NCCL[6], Intel oneCCL[7], 集成了针对不同拓扑结构的 AllReduce 算法, 例如环状的 Ring AllReduce [112]、树状的 Tree-based AllReduce [113]. 新一代高性能并行智能训练系统[8][9] 往往部署了多维通信网络拓扑, 同一个 PCIe Switch 或 NVLink 互连的 GPU 之间、单台服务器内 NUMA 结点之间、服务器之间的通信带宽各不相同, 而常用的 Ring AllReduce 算法难以充分利用多维网络的通信带宽资源. 考虑到结点间带宽一般相对较低, Goyal 等 [36] 将通信过程拆分为结点内 Reduce、结点间 AllReduce 和结点内 Broadcast 三个阶段, 减少了跨结点的通信负载并提高了可扩展性. 类似的, BlueConnect [114]、PLink [115]、Blink [57] 等系统针对多维通信拓扑设计了层次式集合通信算法, 以充分利用结点内的高带宽. Themis [116] 进一步对具备多维通信网络拓扑的并行智能训练系统设计了带宽感知的通信调度算法, 通过调整集合通信中数据块的传输顺序, 提高了系统整体的带宽利用率.

压缩梯度的集合通信. 尽管梯度压缩技术减少了通信量, 但是压缩后的梯度往往无法进行加和操作, 这意味着压缩后的梯度无法执行 AllReduce. 一些研究人员 [106, 108, 110, 117, 118] 注意到传统梯度压缩技术的这一问题, 并开发了支持 AllReduce 操作的梯度压缩技术. 针对梯度稀疏化方法, ScaleCom [106]、SketchSGD [117] 以及基于贝叶斯先验的梯度采样算法 [118] 使用结点间一致的稀疏梯度索引来允许压缩梯度执行 AllReduce. OmniReduce [119] 设计了一种支持块式稀疏化数据 AllReduce 操作的通信系统. GradiVeQ [108]、PowerSGD [110] 将梯度分解为可以直接相加的小矩阵, 其中 PowerSGD 方法得到了 PyTorch 的良好支持.

4.3.4 通信 – 计算调度方法

由于深度学习模型的梯度是从输出层到输入层反向逐层计算的, 当模型的部分层已经计算出梯度, 允许通信时, 部分层仍在执行梯度计算, 因此可以通过通信与计算重叠的方法隐藏部分通信开销. Wait-free back-propagation (WFBP) 技术 [120, 121] 利用这一点将整个模型的梯度分块, 并将计算好的梯度块立即通信, 减少了部分通信时间. 但是, 同时应用 WFBP 技术和梯度压缩技术时会引起 GPU 资源争用, 反而效果不好 [122]. 考虑到这一点, 针对 WFBP 技术与梯度稀疏化相结合的场景, OMGS-WFBP 对模型训练的迭代时间进行建模, 计算出最优的梯度计算 – 稀疏化 – 通信模式.

6) NCCL. Github, 2022. https://github.com/NVIDIA/nccl.
7) Intel oneCCL. https://github.com/oneapi-src/oneCCL.
8) Cloud TPU. https://cloud.google.com/tpu.
9) NVIDIA DGX. https://www.nvidia.com/en-us/data-center/dgx-systems/.

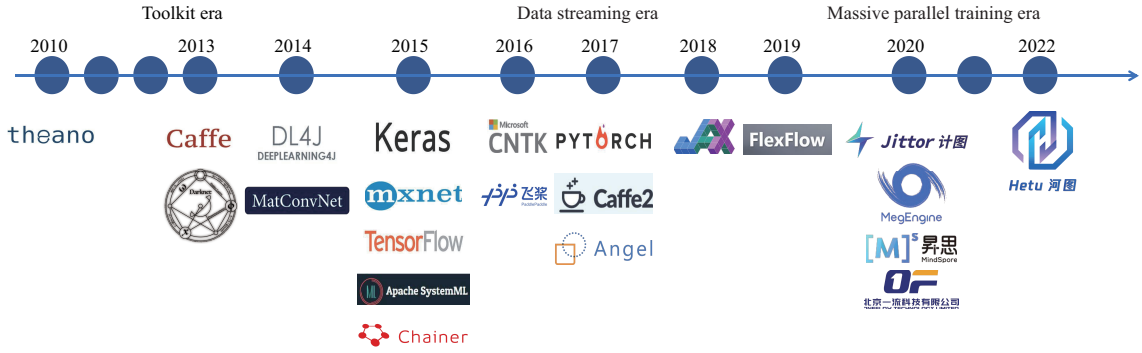

图 6 (网络版彩图) 深度学习框架的发展

Figure 6 (Color online) Development of deep learning frameworks

为了进一步隐藏通信开销, TicTac [123] 提出了通信优先调度技术, 其在参数服务器中将通信队列从先进先出队列改为了优先级队列, 通信的优先级则以前向计算顺序决定, 有着更高优先级的通信提前完成也能提早开始下一轮迭代的前向计算, 从而使得较低优先级的通信操作可以和部分前向计算重叠. 在此技术的基础上, ByteScheduler [124] 将通信调度算法与深度学习计算框架解耦, 并将通信架构从参数服务器拓展至 AllReduce, 同时使用张量分割技术以更细的粒度进行调度, 让通信操作可以更多地被前向计算掩盖, 从而实现整体训练效率的提升. 在后续的工作中, EmbRace [125] 则将通信调度拓展至稀疏张量的每一行, 在有着稀疏梯度的自然语言处理模型中取得更佳的通信计算重叠效果.

5 并行智能训练框架软件

智能训练框架为智能模型的开发提供编程接口和运行环境, 对于深度学习发展具有重要的推动作用. 然而, 随着训练数据和模型参数规模的增长, 智能训练需要的计算设备规模越来越大, 并行智能训练方式也越来越复杂, 传统的 TensorFlow, PyTorch 等基础深度学习框架已经不能满足大规模智能模型的开发和训练需求. 因此, 数据并行训练框架和混合并行训练框架先后被提出, 通过对并行智能训练方法进行封装, 抽象出简单易用的用户接口, 降低并行智能训练的编程难度, 进一步推动深度学习技术的发展.

5.1 基础深度学习框架

深度学习框架为智能计算模型的构建、训练、评估等开发流程提供了基础的编程接口和运行环境. 图 6 展示了深度学习框架随着深度学习的发展从工具包时代到数据流时代再到大规模并行训练时代的发展历程. 2010 年, 首个深度学习框架 Theano [126] 正式发布, 开启了深度学习框架研发的热潮. 目前, 市场上已经出现了大量的深度学习框架. 从用户规模来看, 国外开源深度学习框架 PyTorch [127] 和 TensorFlow [128] 的用户规模遥遥领先, 并且 PyTorch 近年已经实现了对 TensorFlow 的反超, 成为目前最受欢迎的深度学习框架.

5.2 数据并行训练框架

在基础深度学习框架的基础之上, 为了加速模型训练过程, 研究者们开发了数个易用高效的数据并行训练框架. 按照数据并行训练进程之间的参数聚合实现方式, 数据并行训练框架也可以分为中心化和去中心化两个类别. 在中心化的实现方法上, TF-PS [5] 使用 gRPC 通信框架在 TensorFlow 深度

```python
import horovod.torch as hvd
# Initialize Horovod
hvd.init()
...
# Set Horovod optimizer
optimizer = hvd.DistributedOptimizer()
...
# Broadcast parameters
hvd.broadcast_parameters(model.state_dict())
# Do train
for data, target in dataloder:
   optimizer.zero_grad()
   output = model(data)
   loss = F.nll_loss(output, target)
   loss.backward()
   optimizer.step()
```

(a)

```python
import bagua.torch_api as bagua
# Initialize Bagua
bagua.init_process_group()
...
# Transform to Bagua wrapper
from bagua.algorithms.gradient_allreduce \
    import GradientAllReduceAlgorithm
model = model.with_bagua(
    GradientAllReduceAlgorithm())
# Do train
for data, target in dataloder:
   optimizer.zero_grad()
   output = model(data)
   loss = F.nll_loss(output, target)
   loss.backward()
   optimizer.step()
```

(b)

```python
import Merak
# Init parallel degrees
dp, tp, pp = 2, 2, 2
Merak.init(dp, tp, pp)
# Set Merak training args
args=Merak.MerakArguments()
...
# Set Merak trainer
trainer=Merak.MerakTrainer(
    args=args,
    model=...,
    train_data=...,
    eval_data=...,
)
# Do train
trainer.train()
```

(c)

图 7 (网络版彩图) (a) Horovod, (b) Bagua 和 (c) Merak 并行训练框架应用于 PyTorch 训练脚本的代码示意图

Figure 7 (Color online) Code examples of (a) Horovod, (b) Bagua, and (c) Merak parallel training frameworks applied on PyTorch scripts

学习框架上实现了参数服务器架构, 成为了 TensorFlow 默认的数据并行方案. 在去中心化实现方法上, Horovod [4] 是一个兼容多个深度学习框架的数据并行训练框架, 其独立的通信控制模块可以支持多个计算框架与通信库, 带来较为统一的使用方法. Horovod 还采用了能够提高带宽利用率的张量融合 (tensor fusion) 技术和 Ring AllReduce 算法, 取得了优秀的通信效率以及训练规模拓展率. PyTorch DDP [129] 是 PyTorch 深度学习框架默认的数据并行解决方案, 其能更好地利用计算框架特性进行模型计算图遍历, 设立更为准确的梯度桶 (gradient bucket) 通信融合方法, 以避免通信死锁和降低通信控制的开销.

BytePS [130] 则将中心化和去中心化方法的优势相结合, 首先 BytePS 将常见的 CPU-GPU 异构训练集群中空闲的 CPU 资源用于增加参数服务器数量从而提升通信效率, 随后 BytePS 利用结点内的高带宽使用去中心化的通信方法进一步加速通信, 最后 BytePS 将 CPU 上的 server 更改为仅进行梯度累加, 将运算较为复杂的参数更新转移至 GPU worker 之中, 从而提升数据并行效率. 同样支持中心化和去中心化方法的 Bagua [131] 则侧重于支撑更多的通信算法, 包括同步与异步、低精度与高精度、全连接与非全连接、中心化与去中心化等各式算法 [87, 132~134], 使得数据并行训练可以适配更多的系统条件和应用场景.

目前常用的数据并行训练框架对用户较为友好, 如图 7(a) 和 (b) 所示, 以 Horovod 和 Bagua 为例, 用户既不需要对训练脚本进行侵入式的修改, 也不需要重新进行模型的定义. 通常而言, 用户仅需建立多进程之间的通信环境, 在模型反向计算时插入梯度聚合操作 (Horovod 对 PyTorch 的优化器进行封装, Bagua 则直接对模型进行封装), 以及统一模型的初始化过程, 即可成功使用开源社区中大量的单机训练脚本进行高效的数据并行训练.

5.3 混合并行训练框架

随着并行智能训练的高速发展, 多计算设备的混合并行训练成为主要的高效训练方法. 如图 6 所示, 近年来深度学习框架的开发进入大规模并行训练时代, 开发者愈发重视对各项并行计算技术的支撑, 部分框架在设计之初就将并行计算作为重要的基础模块 [135~138]. 然而, 较新的深度学习框架在功能完整性以及社区活跃度等方面与目前最流行的深度学习框架 (PyTorch 和 TensorFlow) 相比仍有差

表 2 基于 PyTorch 与 TensorFlow 的代表性混合并行训练框架对比 a)

Table 2 Comparison on representative hybrid parallel training frameworks based on PyTorch and TensorFlow

Name	DNN framework	Available parallelisms				Open source	Details
		DP	PP	TP	EP		
DeepSpeed [44, 54, 80]	PyTorch	✓	✓*	✓*	✓*	✓	ZeRO-powerd DP, 1-bit optimizers
Megatron-LM [43, 83]		✓	✓*	✓*	–	✓	Interleved pipeline schedule, sequence parallelism
Colossal-AI [139]		✓	✓*	✓*	–	✓	2D, 2.5D, and 3D TP
SageMaker [140]		✓	✓	✓*	–	–	Module-server design for PP
Merak [60]		✓	✓	✓	–	✓	User-friendly API, fast runtime engine
FasterMoE [45]		✓	–	–	✓*	✓	Dynamic shadowing strategy for expert
Tutel [42]		✓	–	✓*	✓*	✓	Adaptive TP and DP parallelism switching
Whale [46]	TensorFlow	✓	✓	✓	–	✓	Hardware-aware load balancing algorithm
Alpa [68]		✓	✓	✓	–	✓	Hierarchical algorithm for auto parallelism
DAPPLE [49]		✓	✓*	–	–	✓	Hybid load balancing parallelization planner
Mesh-TensorFlow [141]		✓*	–	✓*	–	✓	Distribution strategies formalization

a) * indicates that applying this parallellism requires specific modifications to the model

距, 有一定的使用门槛.

表 2 [42~46, 49, 54, 60, 68, 80, 83, 139~141] 中列出了基于 PyTorch 和 TensorFlow 的代表性混合并行训练框架. 研究者们设计了多种巧妙的实现以提高并行训练的效率, 例如, DeepSpeed [44] 在提出了优化冗余内存技术 ZeRO 以外, 还设计了通信更为高效的优化器 1-bit Adam [142] 和 1-bit LAMB [143], 以及渐进式模型层裁剪调度技术 [144] 等训练推理优化技术; Colossal-AI 则侧重于张量并行训练中更高效的矩阵乘算法, 并设计了使 Transformer 模型能够训练更长数据序列的环形自注意力层 [145], 以及面向并行训练设计了一种更高效的基于数据块的内存交换管理方法 [146]; 而 FasterMoE [45] 在专家并行部分进行了大量优化, 其首先使用动态影子专家策略减轻了设备间负载不均衡的问题, 其次用细粒度的通信计算调度策略重叠通信与计算, 提高了专家层的运行效率, 最后设计了拓扑感知的专家选择策略减少网络拥塞, 从而进一步降低通信开销; Whale [46] 则基于 TensorFlow, 建立了两个高级原语表达 DP, PP 和 TP 并行策略以及由这些策略组成的混合并行策略, 并使用装饰器的形式自动将本地模型计算图转为并行模型计算图, 最后 Whale 提出了基于硬件感知的负载均衡算法以提升并行效率.

混合并行训练框架的使用相较数据并行框架更为复杂, 其要求用户对并行智能训练技术有较深的了解. PyTorch 深度学习框架虽以易用性著称, 然而在基于 PyTorch 深度学习框架的混合并行框架之中, 大部分框架要求用户在定义模型时使用特定的格式或者特定的算子接口, 并将所有模型层置于一个可以依次执行的列表中, 这使得用户在使用开源社区中的单机训练脚本时需要对训练脚本进行较大的修改; SageMaker 宣称可以将流水线并行应用至任意模型, 但是这是一个闭源的并行训练框架, 没有开源代码且仅能在亚马逊的云服务上使用.

6 并行智能训练技术发展趋势

并行智能训练技术和框架软件的快速发展, 有效支撑了人工智能算法不断演进, 本节结合现有研究梳理了并行智能训练技术的发展趋势.

6.1 深度学习编译器是提高并行训练可编程性的重要技术方向

作为大规模智能算法开发的编程接口和运行环境, 并行智能训练框架软件架构和编程接口的可编程性对于提高新型人工智能算法开发效率, 促进并行智能训练框架生态发展具有重要意义. 大规模智能算法开发面临数据规模大、内存开销大以及并行编程难的问题. 例如, GPT-3 [3] 模型的参数量达到 1750 亿, 使用 32 位浮点的模型存储量达到 700 GB, 训练过程的内存需求量为 3~4 TB, 采用单一的数据并行或者模型并行技术已经无法满足大规模智能模型的高效并行计算, 多维度的混合并行计算成为大规模智能模型训练的主流技术. 然而, 一般的人工智能模型开发人员往往不了解并行计算技术, 完成模型设计后如何高效地进行模型并行计算成为了重大的现实挑战 [68]. 因此, 如何合理设计并行智能训练编译器、提供简洁易用的编程接口、实现人工智能模型设计与高性能并行计算的解耦, 成为并行智能训练的重要发展趋势.

6.2 大规模并行智能训练的可扩展性仍面临巨大技术挑战

可扩展性是传统高性能计算系统的一项重要设计指标, 对于大规模智能训练任务, 由于其计算密集、通信密集、数据密集的特点, 需要采用大量计算设备和结点, 这使得可扩展性的设计需求更具有挑战性. 此外, 智能训练的每轮迭代, 都需要在各个计算设备之间进行模型参数的集合通信. 对于大规模智能模型, 由于并行方式复杂, 集合通信的模式更加多样复杂, 除了数据并行中的 AllReduce 操作, 还需要在模型并行计算任务之间进行 AlltoAll 或者 AllGather. 大量的模型参数、复杂的并行方式和多样的集合通信, 都给大规模并行智能模型训练的扩展性提出了严峻的技术挑战. 例如英伟达公司的 Megatron-LM [43] 将混合并行训练的主要通信开销限制于结点内, 在至多 3072 个 GPU 上获得了接近线性的弱可拓展性. 最后, 智能模型训练同样也是 I/O 密集的应用, 每轮训练迭代前都需要读取一个批量的训练数据集. 当数据并行规模和批量大小较大时, 特别是对于高维的图像等数据, I/O 的效率也会影响智能训练的扩展性能. 因此, 实现大规模并行智能训练任务的可扩展是高性能智能训练系统的重要趋势. 除了需要更高带宽、低延迟的互连网络硬件, 合理的模型划分和混合并行方式选择、集合通信的实现、通信与计算的调度等软件方法也是提高并行智能训练可扩展性的重要方法.

6.3 混合精度计算是提高智能训练效率的重要技术

大规模智能模型训练的计算量巨大, 完成一个模型的训练可能需要几天甚至几个月, 巨大的能耗是训练大规模智能模型的现实挑战. 相比传统科学计算领域的高性能计算应用, 智能训练具有低精度容忍的特点, 因此混合精度计算方法成为了提高智能训练能效的重要技术手段. 除了 IEEE 标准中的 FP32, FP16 等标准低精度数据格式, 谷歌、英伟达等公司也为深度学习提出了 TF32, BF16 等低精度格式, 并被智能处理器广泛使用. 然而, 面向智能训练的混合精度计算理论和方法仍然面临巨大的研究挑战. 不同的精度数据格式具有不同的数值计算精度, 对于单个数值计算, 数据格式与计算精度的关系是固定的, 但是对于大规模智能模型计算过程, 计算精度分析更加复杂困难. 大规模智能模型, 在空间维度包含大量的神经元和神经网络层, 参数量可以达到万亿规模, 它们的计算是相互关联的; 在时间维度, 智能模型的训练是一个不断迭代神经网络前向计算与反向计算的过程, 训练中学习率等超参数的设置并不是固定的, 仍存在较大的随机性和经验性. CPT [147] 据此于训练过程中采用了动态的计算精度, 在提高能效性的同时提高了收敛精度. 因此, 分析和建模混合精度的计算方法对神经网络训练精度的影响, 也是未来研究的发展趋势之一.

6.4 新型快速训练优化算法对于提升并行训练效率具有很大潜力

智能模型训练的目标是通过大量的训练迭代, 实现智能模型在训练数据集下的模型收敛, 最终使智能分类、检测、问答等下游任务的精度或质量达到用户满意的水平. 对于相同的训练数据集和智能模型, 神经网络训练优化算法 (优化器) 的性能决定了智能模型收敛需要计算的迭代步数. 因此, 新型快速训练优化算法能够直接减少并行智能训练的时间. 在大规模并行智能训练中, 需要使用数据并行将数据分散到多个训练任务单元中进行, 导致一个训练的迭代的理论批量大小 (batch size) 会随着数据并行规模的增大而增大, 而大批量训练的收敛性问题一直是神经网络训练优化理论和实践中的难题 [148]. 例如, 在 ImageNet-1K 训练 ResNet50 模型的挑战赛中, 大批量训练难以收敛的问题是制约算力系统扩展的主要瓶颈, 当批量大小超过 64k, ResNet50 模型就很难收敛. 此外, 优化器的改进也可以直接减少训练过程的内存消耗. 针对优化器内存消耗问题, Chen 等 [149] 提出的 Lion 优化器在保证收敛性的同时可以将额外的内存占用减半. 因此, 研究新型的并行智能训练优化算法, 对于解决并行智能训练的规模受限导致的模型不收敛问题, 提升并行智能训练效率具有巨大潜力.

7 银河天璇并行智能训练框架

面向并行智能训练的挑战与发展, 我们提出并设计了银河天璇 (Merak) 并行智能训练框架. 如图 8 所示, Merak 自顶而下由 6 层组成. 首先, Merak 的用户 API 层将与常用的深度学习计算框架对接, 获取用户使用各种深度学习框架定义的模型; 接着 Merak 将使用计算框架的工具将用户定义的模型转为计算图, 而后将不同格式的计算图转换成统一的 Merak 计算图中间表示 (intermediate representation, IR); 同时 Merak 将获取模型训练所需的计算、通信、存储资源, 并将训练资源使用适当的测量工具进行统计, 例如统计各设备的计算能力和设备之间的通信带宽, 并根据统计结果将训练资源抽象为更有利于处理和计算的数据结构; 随后并行策略制定层将根据训练资源以及计算图中间表示, 使用模型划分关键技术, 选择合适的并行策略搜索算法将计算图合理编译并划分至各个设备, 算法搜索空间将包含所有并行智能计算基本方法, 以确保最优的并行策略应用方案; 分布式运行引擎则侧重于进一步优化并行训练的运行时部分, 联合应用内存优化、通信优化等关键技术, 例如对计算和通信操作进行调度从而尽可能地重叠通信与计算操作等, 分布式运行引擎与并行策略制定层将互相递归调用直至整体运行效率最大化; 最后, Merak 将在计算通信实现层与不同计算设备的通信库和计算库进行适配, 实现计算图在不同体系结构下的高效执行.

银河天璇并行智能训练框架目前已实现 1.0 版本, 并在 GitHub, GitLink 等主流平台开源[10]. 当前版本主要适配了 PyTorch 框架, 针对包含数据并行、流水线并行以及张量并行的混合并行训练, 在模型训练计算图获取和并行训练策略制订层, Merak 实现了模型设计与并行训练的解耦, 降低大规模并行智能训练开发难度. 为此 Merak 设计了一系列的代理算子 (proxy operator), 避免模型在定义阶段消耗大量内存资源, 并且使得 torch.fx [150] 工具可以快速高效地获得完整模型计算图; 此外, Merak 设计了一个可以将任意的模型计算图划分成数个可以序列运行的子图的计算图划分算法, 对模型划分进行了优化, 并使用 AutoPipe [61] 搜索一个较佳的算子间模型划分策略. Merak 目前兼容 Transformers 开源模型库 [151] 及 PyTorch 原生模型, 支持 GPT, BERT, T5, ViT, Swin-Transformer 等大规模模型的自动混合并行训练, 如图 7(c) 所示, Merak 的 API 接口简明易用, 在单机训练脚本的基础上, 仅需增加几行代码即可实现高效的混合并行训练.

10) https://github.com/HPDL-Group/Merak, https://www.gitlink.org.cn/HPDL-Group/Merak.

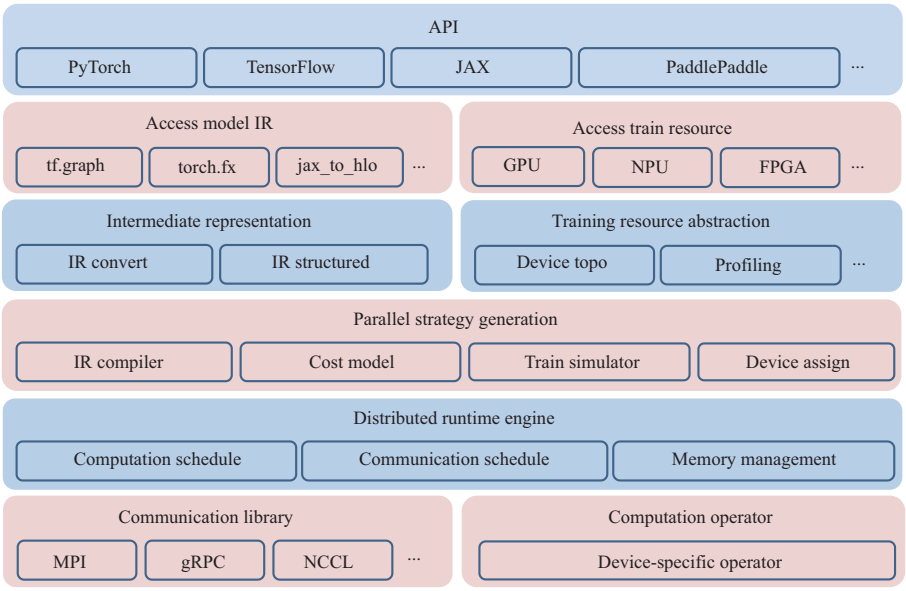

图 8　(网络版彩图) 银河天璇并行智能训练框架架构

Figure 8　(Color online) Architecture of the proposed Merak parallel DNN training framework

在当前版本中, 我们针对多个并行训练关键问题进行了性能优化方案设计, 并在 Merak 分布式运行引擎层中进行了实现. 首先, 在流水线并行中, 我们观察到使用常用的流水线方法调度 4 个流水级训练 4 份 micro-batch 时, 流水线的设备空闲时间至多可达 43%. 据此我们提出了一种转移关键路径的流水线调度方法 (Merak shifted critical path schedule), 相对于 1F1B 调度方法 [49], 能够去除冗余的重计算操作, 提前进行部分重计算操作, 同时通过调整部分 micro-batch 的计算操作顺序, 将大部分流水线关键路径转移到较低的流水级之上, 从而获得一种更高效的流水线调度方法.

其次, 在流水线并行的重计算方面, 现有的实现方法仅提供开启和关闭选项, 我们发现在 RTX 3090 GPU 上训练有 25 亿参数的 GPT 模型时, 开启重计算技术会使得 GPU 存在高达 11 GB 的内存空余 (此时必须开启重计算技术以避免内存溢出). 因此我们根据流水线并行中不同深度的流水级有着不同的存储压力这一特点, 设计了基于流水级感知的重计算技术 (Merak stage-aware recompution), 该技术基于流水线深度, 更细粒度地使用重计算技术, 最大化利用 GPU 空闲内存, 降低重计算技术的额外开销以提高并行训练的效率.

最后, 在张量并行中, 计算过程包含大量的通信操作, 然而现有张量并行的实现中通信操作均为阻塞的 [43], 降低了设备的工作效率, 例如在训练 14 亿参数的 GPT-1.4B 时, 通信时间占比将高达 52.5%. 为此我们提出了张量并行的次级流水化技术 (Merak sub-pipelined TP), 在张量并行内将数据进一步分为独立的子数据, 子数据流水式地执行通信和计算操作, 相较于计算通信互为阻塞的情况, 大量的通信和计算操作可以互相重叠, 从而提高了张量并行的计算效率.

银河天璇并行智能训练框架除了支持通用 GPU 计算集群, 还能够支持国产自主 FT CPU 和 MT-3000 加速器等处理器平台, 在国家超级计算天津中心、国家超级计算长沙中心等高性能计算系统部署应用, 为我国天河超算高效支持大型智能模型训练和推理提供了智能软件框架支撑.

8 总结

人工智能技术发展对于国家经济和国防建设都具有重要意义. 并行智能训练技术和框架软件, 向下依托高性能智能训练硬件系统, 向上支撑人工智能算法开发和运行, 是人工智能技术持续发展的关键, 是智能训练软件生态建设的核心. 本文梳理了并行智能训练的基本模式和关键技术, 以及并行智能训练框架的发展现状, 总结了国内外相关研究工作和最新研究进展, 分析了并行智能训练发展面临的主要技术挑战, 以及作者团队在并行智能训练框架方面的研究成果和发展思路. 随着人工智能算法的快速演进, 作为底层系统支撑技术, 并行智能训练将面临更多严峻挑战, 具有重要的研究价值和广泛的应用前景.

参考文献

1 He K M, Zhang X Y, Ren S Q, et al. Deep residual learning for image recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, 2016. 770–778

2 Russakovsky O, Deng J, Su H, et al. ImageNet large scale visual recognition challenge. Int J Comput Vis, 2015, 115: 211–252

3 Brown T B, Mann B, Ryder N, et al. Language models are few-shot learners. In: Proceedings of the 34th International Conference on Neural Information Processing Systems, Red Hook, 2020. 1877–1901

4 Sergeev A, Balso M D. Horovod: fast and easy distributed deep learning in TensorFlow. 2018. ArXiv:1802.05799

5 Li M, Andersen D G, Park J W, et al. Scaling distributed machine learning with the parameter server. In: Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, Broomfield, 2014. 583–598

6 Devlin J, Chang M W, Lee K, et al. BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2019. 4171–4186

7 Radford A, Wu J, Child R, et al. GPT-2. OpenAi Blog, 2022. https://github.com/openai/gpt-2

8 Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, 2017. 6000–6010

9 Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. In: Proceedings of the 3rd International Conference on Learning Representations, San Diego. 2015

10 Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks. Commun ACM, 2017, 60: 84–90

11 Xu H Z, Gao Y, Yu F, et al. End-to-end learning of driving models from large-scale video datasets. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, 2017. 3530–3538

12 Bicer Y, Alizadeh A, Ure N K, et al. Sample efficient interactive end-to-end deep learning for self-driving cars with selective multi-class safe dataset aggregation. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Macau, 2019. 2629–2634

13 Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. In: Proceedings of the 3rd International Conference on Learning Representations, San Diego, 2015

14 Jean S, Cho K, Memisevic R, et al. On using very large target vocabulary for neural machine translation. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, Beijing, 2015. 1–10

15 Sennrich R, Haddow B, Birch A. Neural machine translation of rare words with subword units. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, Berlin, 2016. 1715–1725

16 Gehring J, Auli M, Grangier D, et al. A convolutional encoder model for neural machine translation. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, Vancouver, 2017. 123–135

17 Dai Z H, Yang Z L, Yang Y M, et al. Transformer-XL: attentive language models beyond a fixed-length context. In: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, 2019. 2978–2988

18 Mehta S, Ghazvininejad M, Iyer S, et al. DeLighT: deep and light-weight transformer. In: Proceedings of International

Conference on Learning Representations, 2021
19 Gulati A, Qin J, Chiu C C, et al. Conformer: convolution-augmented transformer for speech recognition. In: Proceedings of the 21st Annual Conference of the International Speech Communication Association, Shanghai, 2020. 5036–5040
20 Xu Q T, Baevski A, Likhomanenko T, et al. Self-training and pre-training are complementary for speech recognition. In: Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Toronto, 2021. 3030–3034
21 He K M, Gkioxari G, Dollár P, et al. Mask R-CNN. In: Proceedings of IEEE International Conference on Computer Vision (ICCV), Venice, 2017. 2980–2988
22 Rajpurkar P, Zhang J, Lopyrev K, et al. SQuAD: 100000+ questions for machine comprehension of text. In: Proceedings of Conference on Empirical Methods in Natural Language Processing, Austin, 2016. 2383–2392
23 Karras T, Laine S, Aila T. A style-based generator architecture for generative adversarial networks. In: Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, 2019. 4396–4405
24 Raffel C, Shazeer N, Roberts A, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. J Machine Learning Res, 2020, 21: 5485–5551
25 Smith S, Patwary M, Norick B, et al. Using DeepSpeed and Megatron to train megatron-turing NLG 530B, a large-scale generative language model. 2022. ArXiv:2201.11990
26 Zhai X H, Kolesnikov A, Houlsby N, et al. Scaling vision transformers. In: Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, 2022. 1204–1213
27 Abu-El-Haija S, Kothari N, Lee J, et al. YouTube-8M: a large-scale video classification benchmark. 2016. ArXiv:1609.08675
28 Ben-Nun T, Hoefler T. Demystifying parallel and distributed deep learning: an in-depth concurrency analysis. ACM Comput Surv, 2019, 52: 1–43
29 Dean J, Corrado G, Monga R, et al. Large scale distributed deep networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems, Lake Tahoe, 2012. 1223–1231
30 Chilimbi T M, Suzue Y, Apacible J, et al. Project adam: building an efficient and scalable deep learning training system. In: Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation, Broomfield, 2014. 571–582
31 Jouppi N P, Young C, Patil N, et al. In-datacenter performance analysis of a tensor processing unit. In: Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA), Toronto, 2017. 1–12
32 Zhang S J, Du Z D, Zhang L, et al. Cambricon-X: an accelerator for sparse neural networks. In: Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, 2016. 1–12
33 Ouyang J, Du X L, Ma Y, et al. 3.3 Kunlun: a 14nm high-performance AI processor for diversified workloads. In: Proceedings of IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, 2021. 50–51
34 Lu K, Wang Y H, Guo Y, et al. MT-3000: a heterogeneous multi-zone processor for HPC. CCF Trans HPC, 2022, 4: 150–164
35 Mattson P, Cheng C, Diamos G, et al. MLPerf training benchmark. In: Proceedings of Machine Learning and Systems, Austin, 2020
36 Goyal P, Dollár P, Girshick R, et al. Accurate, large minibatch SGD: training ImageNet in 1 hour. 2017. ArXiv:1706.02677
37 Kurth T, Treichler S, Romero J, et al. Exascale deep learning for climate analytics. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, Dallas, 2018. 1–12
38 Hoffmann J, Borgeaud S, Mensch A, et al. Training compute-optimal large language models. 2022. ArXiv:2203.15556
39 Du N, Huang Y P, Dai A M, et al. GLaM: efficient scaling of language models with mixture-of-experts. In: Proceedings of the 39th International Conference on Machine Learning, Baltimore, 2022. 5547–5569
40 Narayanan D, Harlap A, Phanishayee A, et al. PipeDream: generalized pipeline parallelism for DNN training. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles, Huntsville, 2019. 1–15
41 Shoeybi M, Patwary M, Puri R, et al. Megatron-LM: training multi-billion parameter language models using model parallelism. 2019. ArXiv:1909.08053

42 Hwang C, Cui W, Xiong Y F, et al. Tutel: adaptive mixture-of-experts at scale. 2022. ArXiv:2206.03382

43 Narayanan D, Shoeybi M, Casper J, et al. Efficient large-scale language model training on GPU clusters using Megatron-LM. In: Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis, St. Louis, 2021. 1–15

44 Rasley J, Rajbhandari S, Ruwase O, et al. DeepSpeed: system optimizations enable training deep learning models with over 100 billion parameters. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Virtual Event, 2020. 3505–3506

45 He J A, Zhai J D, Antunes T, et al. 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, Seoul, 2022. 120–134

46 Jia X Y, Jiang L, Wang A, et al. Whale: efficient giant model training over heterogeneous GPUs. In: Proceedings of USENIX Annual Technical Conference, Carlsbad, 2022. 673–688

47 Huang Y P, Cheng Y L, Bapna A, et al. GPipe: efficient training of giant neural networks using pipeline parallelism. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems, Vancouver, 2019. 103–112

48 Ye X Y, Lai Z Q, Li S W, et al. Hippie: a data-paralleled pipeline approach to improve memory-efficiency and scalability for large DNN training. In: Proceedings of the 50th International Conference on Parallel Processing, Lemont, 2021. 1–10

49 Fan S Q, Rong Y, Meng C, et al. DAPPLE: a pipelined data parallel approach for training large models. In: Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Virtual Event, 2021. 431–445

50 Li S G, Hoefler T. Chimera: efficiently training large-scale neural networks with bidirectional pipelines. In: Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis, St. Louis, 2021. 1–14

51 Xu Q F, Li S G, Gong C Y, et al. An efficient 2D method for training super-large deep learning models. 2021. ArXiv:2104.05343

52 Wang B X, Xu Q F, Bian Z D, et al. 2.5-dimensional distributed model training. 2021. ArXiv:2105.14500

53 Bian Z D, Xu Q F, Wang B X, et al. Maximizing parallelism in distributed training for huge neural networks. 2021. ArXiv:2105.14450

54 Rajbhandari S, Li C L, Yao Z W, et al. DeepSpeed-MoE: advancing mixture-of-experts inference and training to power next-generation AI scale. In: Proceedings of the 39th International Conference on Machine Learning, Baltimore, 2022. 18332–18346

55 Fedus W, Zoph B, Shazeer N. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. J Machine Learning Res, 2022, 23: 5232–5270

56 Patarasuk P, Yuan X. Bandwidth efficient all-reduce operation on tree topologies. In: Proceedings of IEEE International Parallel and Distributed Processing Symposium, Long Beach, 2007. 1–8

57 Wang G H, Venkataraman S, Phanishayee, et al. A blink: fast and generic collectives for distributed ML. In: Proceedings of Machine Learning and Systems, Austin, 2020

58 Lepikhin D, Lee H J, Xu Y Z, et al. Gshard: scaling giant models with conditional computation and automatic sharding. In: Proceedings of the 9th International Conference on Learning Representations, Virtual Event, 2021

59 Duan Y, Lai Z, Li S, et al. HPH: hybrid parallelism on heterogeneous clusters for accelerating large-scale DNNs training. In: Proceedings of IEEE International Conference on Cluster Computing (CLUSTER), Heidelberg, 2022. 313–323

60 Lai Z, Li S, Tang X, et al. Merak: an efficient distributed DNN training framework with automated 3D parallelism for giant foundation models. IEEE Trans Parallel Distrib Syst, 2023, 34: 1466–1478

61 Liu W J, Lai Z Q, Li S W, et al. AutoPipe: a fast pipeline parallelism approach with balanced partitioning and micro-batch slicing. In: Proceedings of IEEE International Conference on Cluster Computing (CLUSTER), Heidelberg, 2022. 301–312

62 Liang P, Tang Y, Zhang X D, et al. A survey on auto-parallelism of neural networks training. TechRxiv, 2022. doi: 10.36227/techrxiv.19522414.v1

63 Tarnawski J, Narayanan D, Phanishayee A. Piper: multidimensional planner for DNN parallelization. In: Proceedings of Neural Information Processing Systems (NeurIPS), 2021. 24829–24840
64 Eliad S, Hakimi I, Jagger A D, et al. Fine-tuning giant neural networks on commodity hardware with automatic pipeline model parallelism. In: Proceedings of USENIX Annual Technical Conference (USENIX ATC 21), 2021. 381–396
65 Jia Z H, Lin S N, Qi C R, et al. Exploring hidden dimensions in parallelizing convolutional neural networks. In: Proceedings of the 35th International Conference on Machine Learning, Stockholmsmässan, 2018. 2274–2283
66 Cai Z, Yan X, Ma K, et al. TensorOpt: exploring the tradeoffs in distributed DNN training with auto-parallelism. IEEE Trans Parallel Distrib Syst, 2022, 33: 1967–1981
67 Jia Z H, Zaharia M, Aiken A. Beyond data and model parallelism for deep neural networks. In: Proceedings of the 35th International Conference on Machine Learning, Stanford, 2019
68 Zheng L M, Li Z H, Zhuang Y H, et al. Alpa: automating inter- and intra-operator parallelism for distributed deep learning. In: Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), Carlsbad, 2022. 559–578
69 Unger C, Jia Z H, Wu W, et al. Unity: accelerating DNN training through joint optimization of algebraic transformations and parallelization. In: Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), Carlsbad, 2022. 267–284
70 Rhu M, Cimelshein N, Clemons J, et al. vDNN: virtualized deep neural networks for scalable, memory-efficient neural network design. In: Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, 2016. 1–13
71 Chen X M, Chen D Z, Hu X B. moDNN: memory optimal DNN training on GPUs. In: Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, 2018. 13–18
72 Huang C C, Jin G, Li J Y. SwapAdvisor: pushing deep learning beyond the GPU memory limit via smart swapping. In: Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, 2020. 1341–1355
73 Chen T Q, Xu B, Zhang C Y, et al. Training deep nets with sublinear memory cost. 2016. ArXiv:1604.06174
74 Jain P, Jain A, Nrusimha A, et al. Checkmate: breaking the memory wall with optimal tensor rematerialization. In: Proceedings of the 3rd Conference Machine Learning and Systems, Austin, 2020
75 Kirisame M, Lyubomirsky S, Haan A, et al. Dynamic tensor rematerialization. In: Proceedings of the 9th International Conference on Learning Representations, Virtual Event, 2021
76 Wang L N, Ye J M, Zhao Y Y, et al. SuperNeurons: dynamic GPU memory management for training deep neural networks. In: Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Vienna, 2018. 41–53
77 Peng X, Shi X, Dai H, et al. Capuchin: tensor-based GPU memory management for deep learning. In: Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, 2020. 891–905
78 Tang Y, Wang C Y, Zhang Y F, et al. DELTA: dynamically optimizing GPU memory beyond tensor recomputation. 2022. ArXiv:2203.15980
79 Zhou Q, Wang H, Yu X, et al. MPress: democratizing billion-scale model training on multi-GPU servers via memory-saving inter-operator parallelism. In: Proceedings of IEEE International Symposium on High-Performance Computer Architecture (HPCA), Montreal, 2023. 556–569
80 Rajbhandari S, Rasley J, Ruwase O, et al. ZeRO: memory optimizations toward training trillion parameter models. In: Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis, Atlanta, 2020. 1–16
81 Ren J, Rajbhandari S, Aminabadi R Y, et al. ZeRO-Offload: democratizing billion-scale model training. In: Proceedings of USENIX Annual Technical Conference, 2021. 551–564
82 Rajbhandari S, Ruwase O, Rasley J, et al. ZeRO-Infinity: breaking the GPU memory wall for extreme scale deep learning. In: Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis, St. Louis, 2021. 1–14
83 Korthikanti V, Casper J, Lym S, et al. Reducing activation recomputation in large transformer models. 2022.

ArXiv:2205.05198
84 McDonald R, Hall K, Mann G. Distributed training strategies for the structured perceptron. In: Proceedings of Annual Conference of the North American Chapter of the Association for Computational Linguistics, Los Angeles, 2010. 456–464
85 Agarwal A, Duchi J C. Distributed delayed stochastic optimization. In: Proceedings of the 24th International Conference on Neural Information Processing Systems, Granada, 2011. 873–881
86 Recht B, Re C, Wright S, et al. HOGWILD!: a lock-free approach to parallelizing stochastic gradient descent. In: Proceedings of the 24th International Conference on Neural Information Processing Systems, Granada, 2011. 693–701
87 Lian X R, Zhang C, Zhang H, et al. Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, 2017. 5336–5346
88 Ho Q R, Cipar J, Cui H G, et al. More effective distributed ML via a stale synchronous parallel parameter server. In: Proceedings of the 26th International Conference on Neural Information Processing Systems, Lake Tahoe, 2013. 1223–1231
89 Zhao X, An A J, Liu J F, et al. Dynamic stale synchronous parallel distributed training for deep learning. In: Proceedings of the 39th International Conference on Distributed Computing Systems (ICDCS), Dallas, 2019: 1507–1517
90 Seide F, Fu H, Droppo J, et al. 1-Bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs. In: Proceedings of the 15th Annual Conference of the International Speech Communication Association, 2014
91 Alistarh D, Grubic D, Li J, et al. QSGD: communication-efficient SGD via gradient quantization and encoding. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, 2017. 1707–1718
92 Wen W, Xu C, Yan F, et al. TernGrad: ternary gradients to reduce communication in distributed deep learning. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, 2017. 1508–1518
93 Bernstein J, Wang Y X, Azizzadenesheli K, et al. SignSGD: compressed optimisation for non-convex problems. In: Proceedings of the 35th International Conference on Machine Learning, Stockholm, 2018. 560–569
94 Karimireddy S P, Rebjock Q, Stich S, et al. Error feedback fixes signSGD and other gradient compression schemes. In: Proceedings of the 36th International Conference on Machine Learning, Long Beach, 2019. 3252–3261
95 Lim H, Andersen D G, Kaminsky M. 3LC: lightweight and effective traffic compression for distributed machine learning. In: Proceedings of Machine Learning and Systems, Stanford, 2019. 53–64
96 Jiang J W, Fu F C, Yang T, et al. SketchML: accelerating distributed machine learning with data sketches. In: Proceedings of International Conference on Management of Data, Houston, 2018. 1269–1284
97 Fu F C, Hu Y Z, He Y H, et al. Don't waste your bits! Squeeze activations and gradients for deep neural networks via tinyscript. In: Proceedings of the 37th International Conference on Machine Learning, 2020. 3304–3314
98 Faghri F, Tabrizian I, Markov I, et al. Adaptive gradient quantization for data-parallel SGD. In: Proceedings of the 34th International Conference on Neural Information Processing Systems, Vancouver, 2020. 3174–3185
99 Bai Y H, Li C, Zhou Q, et al. Gradient compression supercharged high-performance data parallel DNN training. In: Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event, 2021. 359–375
100 Ström N. Scalable distributed DNN training using commodity GPU cloud computing. In: Proceedings of Interspeech, 2015
101 Stich S U, Cordonnier J-B, Jaggi M. Sparsified SGD with memory. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, Montréal, 2018. 4452–4463
102 Lin Y J, Han S, Mao H Z, et al. Deep gradient compression: reducing the communication bandwidth for distributed training. In: Proceedings of the 6th International Conference on Learning Representations, Vancouver, 2018
103 Wangni J Q, Wang J L, Liu J, et al. Gradient sparsification for communication-efficient distributed optimization. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, Montréal, 2018.

1306–1316
104 Renggli C, Ashkboos S, Aghagolzadeh M, et al. SparCML: high-performance sparse communication for machine learning. In: Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis, Colorado, 2019. 1–15
105 Shi S H, Zhao K Y, Wang Q, et al. A convergence analysis of distributed SGD with communication-efficient gradient sparsification. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence, Macao, 2019. 3411–3417
106 Chen C Y, Ni J M, Lu S T, et al. ScaleCom: scalable sparsified gradient compression for communication-efficient distributed training. In: Proceedings of the 34th International Conference on Neural Information Processing Systems, Vancouver, 2020. 13551–13563
107 Zhang Z R, Wang C L. MIPD: an adaptive gradient sparsification framework for distributed DNNs training. IEEE Trans Parallel Distrib Syst, 2022, 33: 3053–3066
108 Yu M C, Lin Z F, Narra K, et al. GradiVeQ: vector quantization for bandwidth-efficient gradient aggregation in distributed CNN training. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, Montréal, 2018. 5129–5139
109 Wang H Y, Sievert S, Charles Z, et al. ATOMO: communication-efficient learning via atomic sparsification. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, Montréal, 2018. 9872–9883
110 Vogels T, Karimireddy P S, Jaggi M. PowerSGD: practical low-rank gradient compression for distributed optimization. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems, Vancouver, 2019. 14269–14278
111 Agarwal S, Wang H Y, Lee K, et al. Accordion: adaptive gradient communication via critical learning regime identification. In: Proceedings of Machine Learning and Systems, 2021. 55–80
112 Chan E, van De Geijn R, Gropp W, et al. Collective communication on architectures that support simultaneous communication over multiple links. In: Proceedings of the 11th ACM SIGPLAN symposium on Principles and practice of parallel programming, New York, 2006. 2–11
113 Patarasuk P, Yuan X. Bandwidth optimal all-reduce algorithms for clusters of workstations. J Parallel Distributed Computing, 2009, 69: 117–124
114 Cho M, Finkler U, Kung D. BlueConnect: decomposing all-reduce for deep learning on heterogeneous network hierarchy. In: Proceedings of Machine Learning and Systems, Stanford, 2019
115 Luo L, West P, Krishnamurthy A, et al. PLink: discovering and exploiting locality for accelerated distributed training on the public cloud. In: Proceedings of Machine Learning and Systems, Austin, 2020
116 Rashidi S, Won W, Srinivasan S, et al. Themis: a network bandwidth-aware collective scheduling policy for distributed training of DL models. In: Proceedings of the 49th Annual International Symposium on Computer Architecture, New York, 2022. 581–596
117 Ivkin N, Rothchild D, Ullah E, et al. Communication-efficient distributed SGD with sketching. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems, Vancouver, 2019. 13142–13152
118 Song L Y H, Zhao K, Pan P, et al. Communication efficient SGD via gradient sampling with Bayes prior. In: Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, 2021. 12060–12069
119 Fei J W, Ho C Y, Sahu A N, et al. Efficient sparse collective communication and its application to accelerate distributed deep learning. In: Proceedings of ACM Special Interest Group on Data Communication, Virtual Event, 2021. 676–691
120 Zhang H, Zheng Z Y, Xu S Z, et al. Poseidon: an efficient communication architecture for distributed deep learning on GPU clusters. In: Proceedings of USENIX Annual Technical Conference (USENIX ATC 17), Santa Clara, 2017. 181–193
121 Shi S H, Chu X W, Li B. MG-WFBP: efficient data communication for distributed synchronous SGD algorithms. In: Proceedings of IEEE Conference on Computer Communications, Paris, 2019. 172–180
122 Agarwal S, Wang H Y, Venkataraman S, et al. On the utility of gradient compression in distributed training systems. In: Proceedings of Machine Learning and Systems, Santa Clara, 2022

123 Hashemi S H, Jyothi S A, Campbell R H. TicTac: accelerating distributed deep learning with communication scheduling. In: Proceedings of Machine Learning and Systems, Stanford, 2019. 418–430
124 Peng Y H, Zhu Y B, Chen Y R, et al. A generic communication scheduler for distributed DNN training acceleration. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles, Huntsville, 2019. 16–29
125 Li S W, Lai Z Q, Li D S, et al. EmbRace: accelerating sparse communication for distributed training of deep neural networks. In: Proceedings of the 51st International Conference on Parallel Processing, Bordeaux, 2022. 1–11
126 AI-Rfou R, Alain G, Almahairi A, et al. Theano: a Python framework for fast computation of mathematical expressions. 2016. ArXiv:1605.02688
127 Paszke A, Gross S, Massa F, et al. PyTorch: an imperative style, high-performance deep learning library. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems, Vancouver, 2019. 8026–8037
128 Abadi M, Agarwal A, Barham P, et al. TensorFlow: large-scale machine learning on heterogeneous distributed systems. 2016. ArXiv:1603.04467
129 Li S, Zhao Y, Varma R, et al. PyTorch distributed: experiences on accelerating data parallel training. Proc VLDB Endow, 2020, 13: 3005–3018
130 Jiang Y M, Zhu Y B, Lan C, et al. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In: Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation, Virtual Event, 2020. 463–479
131 Gan S D, Jiang J, Yuan B, et al. Bagua: scaling up distributed learning with system relaxations. Proc VLDB Endow, 2021, 15: 804–813
132 Koloskova A, Stich S, Jaggi M. Decentralized stochastic optimization and gossip algorithms with compressed communication. In: Proceedings of the 36th International Conference on Machine Learning. California, 2019. 3478–3487
133 Tang H L, Lian X R, Yan M, et al. D^2: decentralized training over decentralized data. In: Proceedings of the 35th International Conference on Machine Learning. Stockholm, 2018. 4848–4856
134 Sa D C, Feldman M, Ré C, et al. Understanding and optimizing asynchronous low-precision stochastic gradient descent. In: Proceedings of the 44th Annual International Symposium on Computer Architecture, Toronto, 2017, 561–574
135 Lei C. Deep Learning and Practice With MindSpore. Singapore: Springer, 2021. 394
136 Yuan J H, Li X Q, Liu J C, et al. OneFlow: redesign the distributed deep learning framework from scratch. 2021. ArXiv:2110.15032
137 Miao X P, Zhang H L, Shi Y L, et al. HET: scaling out huge embedding model training via cache-enabled distributed framework. Proc VLDB Endow, 2021, 15: 312–320
138 Ao Y L, Wu Z H, Gong W B, et al. End-to-end adaptive distributed training on PaddlePaddle. 2021. ArXiv:2112.02752
139 Bian Z D, Liu H X, Wang B X, et al. Colossal-AI: a unified deep learning system for large-scale parallel training. 2021. ArXiv:2110.14883
140 Karakus C, Huilgol R, Wu F, et al. Amazon SageMaker model parallelism: a general and flexible framework for large model training. 2021. ArXiv:2111.05972
141 Shazeer N, Cheng Y L, Parmar N, et al. Mesh-TensorFlow: deep learning for supercomputers. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, Montréal, 2018. 10435–10444
142 Tang H L, Gan S D, Awan A A, et al. 1-bit adam: communication efficient large-scale training with Adam's convergence speed. In: Proceedings of the 38th International Conference on Machine Learning, Virtual Event, 2021. 10118–10129
143 Li C L, Awan A A, Tang H L, et al. 1-bit LAMB: communication efficient large-scale large-batch training with LAMB's convergence speed. In: Proceedings of the 29th International Conference on High Performance Computing, Data, and Analytics (HiPC), Bengaluru, 2022. 272–281
144 Zhang M J, He Y X. Accelerating training of transformer-based language models with progressive layer dropping. In: Proceedings of the 34th International Conference on Neural Information Processing Systems, Vancouver, 2020. 14011–14023

145 Li S G, Xue F Z, Branwal C, et al. Sequence parallelism: long sequence training from system perspective. In: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, Toronto, 2023. 2391–2404
146 Fang J, Zhu Z, Li S, et al. Parallel training of pre-trained models via chunk-based dynamic memory management. IEEE Trans Parallel Distrib Syst, 2023, 34: 304–315
147 Fu Y G, Guo H, Li M, et al. CPT: efficient deep neural network training via cyclic precision. 2021. ArXiv:2101.09868
148 Keskar N S, Mudigere D, Nocedal J, et al. On large-batch training for deep learning: generalization gap and sharp minima. In: Proceedings of the 5th International Conference on Learning Representations, Toulon, 2017
149 Chen X L, Liang C, Huang D, et al. Symbolic discovery of optimization algorithms. 2023. ArXiv:2302.06675
150 Reed J K, DeVito Z, He H, et al. Torch.fx: practical program capture and transformation for deep learning in Python. 2021. ArXiv:2112.08429
151 Wolf T, DebutL, Sanh V, et al. Transformers: state-of-the-art natural language processing. In: Proceedings of Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Online, 2020. 38–45

Parallel intelligent computing: development and challenges

Kai LU*, Zhiquan LAI, Shengwei LI, Weijie LIU, Keshi GE,
Xicheng LU & Dongsheng LI

National Key Laboratory of Parallel and Distributed Processing, College of Computer, National University of Defense Technology, Changsha 410073*, China*
* Corresponding author. E-mail: kailu@nudt.edu.cn

Abstract The development of artificial intelligence technology with the use of deep learning has gained momentum in recent years, resulting in a considerable increase in the scale of deep learning models and training data, and high-performance computing and artificial intelligence technologies have been continuously and deeply integrated. Parallel intelligent training has become the main method for the efficient training of large-scale deep learning models. In this work, we evaluate the basic methods and key technologies of parallel intelligent training, outline the development status of the parallel intelligent training framework, summarize the challenges and trends in developing the parallel intelligent technology and framework, and propose development schemes for a Merak parallel intelligent training framework.

Keywords intelligent training, high-performance computing, parallel intelligent training, deep learning