计算机科学
COMPUTER SCIENCE

一种面向通用计算设备的自动流水线并行训练框架

钟震宇, 林勇良, 王昊天, 李东闻, 孙羽菲, 张玉志

引用本文

钟震宇, 林勇良, 王昊天, 李东闻, 孙羽菲, 张玉志. 一种面向通用计算设备的自动流水线并行训练框架[J]. 计算机科学, 2024, 51(12): 129-136.

ZHONG Zhenyu, LIN Yongliang, WANG Haotian, LI Dongwen, SUN Yufei, ZHANG Yuzhi. Automatic Pipeline Parallel Training Framework for General-purpose Computing Devices [J]. Computer Science, 2024, 51(12): 129-136.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.231000110

一种面向通用计算设备的自动流水线并行训练框架

钟震宇　林勇良　王昊天　李东闻　孙羽菲　张玉志
南开大学软件学院　天津 300350
(zyzhong@mail.nankai.edu.cn)

摘　要　训练大规模神经网络通常会出现单个计算节点的内存和计算能力不足的情况,需要通过多个节点分布式训练来实现。现有的分布式深度学习框架主要针对特定的硬件环境设计,不能够有效适应各类通用计算设备。为支持大规模深度神经网络的高效训练,实现了一种通用的自动流水线并行分布式训练框架。本框架通过结合基于流水线并行的模型并行策略与神经网络模型自动拆分算法,实现了在包括国内新一代超级计算机在内的通用计算机集群上,对大规模神经网络模型与训练数据进行自动并行化处理和训练,显著减轻单个计算节点的内存和计算压力。该框架无需人工调整,可以自动高效地在多节点分布式环境中部署深度神经网络,不仅适用于超级计算机等高性能计算机集群,还可以部署到其他通用的分布式计算环境中,为大规模神经网络的自动化分布式训练提供支持。

关键词:流水线并行;深度神经网络;超级计算机;MPI;并行计算
中图分类号　TP183

Automatic Pipeline Parallel Training Framework for General-purpose Computing Devices

ZHONG Zhenyu,LIN Yongliang,WANG Haotian,LI Dongwen,SUN Yufei and ZHANG Yuzhi
College of Software,Nankai University,Tianjin 300350,China

Abstract　Training large-scale neural networks usually exceeds the memory and computing capacity of a single computing node, which requires distributed training using multiple nodes. Existing distributed deep learning frameworks are mainly designed for specific hardware environments and cannot effectively adapt to various general-purpose computing devices. To support the efficient training of large-scale deep neural networks,this paper implements a general-purpose automatic pipeline parallel distributed training framework. This framework combines the model parallel strategy based on pipeline parallelism with the algorithm that automatically splits the neural network model,and realizes the automatic parallelization and training of large-scale neural network models and training data on general computer clusters,including the new generation of supercomputers in China,significantly reducing the memory and computing pressure of a single computing node. The framework does not require manual adjustment,and can automatically and efficiently deploy deep neural networks to multi-node distributed environments. It is not only suitable for supercomputers and other high-performance computer clusters,but also can be deployed to other general distributed computing environments,providing support for the automatic distributed training of large-scale neural networks.

Keywords　Pipeline parallelism,Deep neural network,Supercomputer,Message passing interface,Parallel computing

1　引言

在深度学习领域,大规模训练是通向强大模型的有效途径[1-2]。使用大量数据集训练出的大参数量的简单模型的性能远超结构复杂的模型[2-4]。因此,越来越多的大型深度神经网络(DNN)模型被提出。例如,自然语言处理领域的大型预训练模型 GPT-3[1]拥有 1 750 亿个参数。使用海量数据训练大参数量 DNN 模型的任务是计算密集型的:随着训练所需的数据量与计算量的快速增长,一个大型模型需要在大规模计算节点上被连续训练数周甚至数月才能达到最优的推断能力。

近年来,为了提高计算机系统的计算能力和计算效率,研究人员在硬件与软件方面都做出了许多努力。许多大型计算机系统被研究制造出来,这些大型计算机系统往往拥有大量的计算节点以及强大的整体计算能力。但是,单个计算节点的内存空间依然有限,单节点无法承载完整的大型 DNN 模型。因此,只有通过模型并行的方法使用多个计算节点合作承载一个大模型,才能支撑起大型 DNN 模型的训练。现有的适用于大型模型的模型并行训练框架大多是基于英伟达

到稿日期:2023-10-18　返修日期:2024-03-11
基金项目:国家重点研发计划(2021YFB0300104)
This work was supported by the National Key Research and Development Program of China(2021YFB0300104).
通信作者:孙羽菲(yufei_sun@sina.com)

CUDA 架构[5]设计的。即使有少量支持通用计算设备的训练框架出现，也存在着通信效率低、计算不稳定等问题，这极大程度上限制了深度学习在通用计算设备上的实际应用与发展。在本文中，通用计算设备指各种可以应用于异构计算加速的设备，例如 GPGPU，DSP，FPGA 等，尤其指那些不支持英伟达 CUDA 架构[5]、无法利用现有专门面向英伟达 CUDA 架构设计的并行框架的计算设备，包括但不限于通用 OpenCL 设备、天河新一代超算系统中使用的 DSP 设备[6]等。天河新一代超算系统拥有强大的计算能力和大规模的通用计算节点，但是现有的模型并行框架并不能很好地支持超算环境，这使得在国产超级计算机上训练大型 DNN 模型较为困难。

大规模分布式训练主要面临的挑战有两个方面。一是空间开销大。所有 DNN 模型训练的相关数据（激活值、模型参数、梯度、优化器参数及其他临时变量）都必须被计算设备的内存（CPU 内存、GPU 内存等）容纳。当这些数据过大而无法被存放进设备的内存中，就会导致设备内存溢出从而无法进行训练[7-9]。大型计算机虽然性能优异，但单个计算节点的资源仍是有限的，在训练大规模 DNN 模型时也会面临内存溢出的问题。二是数据与模型划分困难。进行分布式训练时，需要将训练数据和模型参数划分到不同的计算节点上。手动划分数据工作量大，不可扩展。同时，模型代码的拆分以及不同模型分块间的梯度传递工作往往由模型的设计者来完成。因此，流水线等模型并行技术的应用仍面临重大挑战和阻碍。

为了应对上述挑战，本文实现了一个适配通用计算设备的、自动拆分 DNN 模型进行流水线并行训练的分布式训练框架。该框架简单易用，用户仅需给出简单的配置输入，无需对训练数据进行手动拆分，也无需对原有 DNN 模型定义进行任何更改，即可在通用大规模计算集群上进行大规模 DNN 模型分布式训练。本框架用户透明的设计模式使得模型设计者可以专注于模型网络本身的设计，无需为解决与分布式并行训练相关的问题花费过多时间与精力。

本文的贡献总结如下：

1）实现了一种自动拆分 DNN 模型的流水线并行策略，使得大型 DNN 模型可以通过流水线方式进行自动模型并行。该策略包括从不同节点提取梯度并传递的核心算法，极大地减轻了大型深度学习模型对内存的压力，减少了分布式训练的空间开销。

2）实现了一种根据用户配置自动拆分训练数据的机制，该机制可兼容多种数据并行算法，进一步提高训练效率，减少用户工作量。

3）构建了一种适用于通用计算设备且高效易用的模型并行训练框架。用户只需对训练代码做少量修改并提供简单的配置信息，即可自动实现数据与模型的拆分和流水线并行训练。本框架支持 TensorFlow 和 PyTorch 等主流深度学习框架。实验结果表明，本框架可以高效且正确地支持大型深度学习训练任务的执行和完成。

2 背景和相关工作

模型并行是一种分布式训练技术，用于解决因模型太大而无法被单个计算设备承载的问题[10]。与数据并行不同，模型并行将一个单一的模型分割到不同的计算设备上，这样一组计算设备可以共同支撑一个大模型。通常一组训练数据会依次流经这组设备的所有节点。由于模型并行涉及对原始模型进行拆分等复杂问题，传统的模型并行算法需要针对不同的模型进行单独的分析与设计，很难有一种通用的解决方案，因此目前成熟的模型并行框架和算法较少。Megatron-LM[10]是由英伟达开发的一个应用于自然语言处理领域的 DNN 模型，其实现了模型并行。该模型使用 PyTorch 实现，利用混合精度训练和英伟达 NCCL 通信库实现 GPU 之间的高效通信。Megatron-LM 在 512 个 GPU 上使用 8 路模型并行和 64 路数据并行的配置，训练出了拥有 83 亿个参数的基于 Transformer[11]结构的模型，成为当时最大的语言模型。但其仍然存在两点不足：一是其对 GPU 的使用不够高效，实际性能只有理论性能的一半左右；二是其与 CUDA 库高度耦合，无法在其他通用计算设备上使用。

GPipe[12]是由 Google 提出的另一种较为成熟的基于流水线并行的模型并行算法。流水线并行方式把 DNN 模型的不同层划分到不同机器上，在同一批次间串行、在不同批次间并行地进行前向和反向计算。这样的好处是可以让模型摆脱单一计算节点内存大小带来的限制，将模型扩大到任意规模，在图像和自然语言处理方面的模型上都取得更好的训练加速效果。但是，GPipe 也存在两点不足：一是其代码不兼容流行的深度学习框架（如 TensorFlow 或 PyTorch 等），需要用户自行修改模型代码，带来了额外的工作量；二是其对计算节点的内存需求较高，且没有自动划分模型分区的机制，需要用户手动指定每个计算节点上的模型分区，才能保证正常训练。

还有一些高级混合并行框架，如由微软开发的 DeepSpeed[13]，其不仅支持基础的模型并行，还引入了 ZeRO 并行[14-16]等高级并行技术，进一步增强了模型并行的能力。FSDP 技术[17]通过将模型参数分发到不同的计算节点上来降低计算节点的存储需求。飞桨深度学习平台[18]运用多种高性能并行训练方法，实现了支持千亿特征、万亿参数、数百节点的开源大规模训练平台，并实现了万亿规模参数模型的实时更新。BPipe[19]技术对节点内存使用进行了进一步的优化。但遗憾的是，这些技术或框架仍然无法在通用计算设备上使用，仅支持英伟达的 GPU 设备。面向通用计算设备的并行训练技术和框架目前还较少，有待进一步研究。

3 面向通用计算设备的自动流水线并行训练框架

3.1 设计思想和原则

本文提出的分布式训练框架以 TensorFlow 和 PyTorch 等流行深度学习框架为基础，使用高性能、跨平台性能良好的 MPI 为通信依赖库。在设计与实现过程中，秉持以下核心理念与原则：

1）面向通用计算设备：实现中不包含任何 CUDA 依赖，因此可以在不支持 CUDA 的通用计算设备上运行，支持 CPU，GPU，DSP 等各种计算设备和加速器。

2）高可扩展性：设计充分考虑在不同规模计算集群上的可扩展性。通过灵活的节点分配方案，可以适应不同规模的计算集群，保证一致的高性能表现。

3）用户友好的编程接口：编程接口的易用性对于框架的广泛应用至关重要。设计简单、直观、一致的 API，有助于开发用户快速上手。无需手动修改模型定义，也无需手动划分数据，采用用户透明的设计模式来发挥框架的强大功能。

4）基于主流深度学习框架：采用 TensorFlow 和 PyTorch 等流行的深度学习框架作为基础，与用户现有的技术无缝集成。这使得用户能够在熟悉的环境中工作为用户提供一种延续的、无缝对接的开发体验。

3.2 框架 API 设计

相较于传统分布式训练需要人工划分数据与拆分模型，本框架仅要求用户提供简单的输入：1）使用深度学习框架 API 定义的 DNN 模型、优化器、损失函数（即原 DNN 模型定义）；2）每个计算节点上要放置的 DNN 模型的层数。

在下列伪代码示例中，用户提供了一个包含输入层（Flatten）、全连接层（Dense）和暂退层（Dropout）的模型，并通过参数设定，将模型划分到 4 个计算节点，每个计算节点容纳两层神经网络（次要变量的声明已略去）。

```
1. import nkflow as nf
2. import tensorflow as tf
3.
4. # 每个计算节点上所要放置的 DNN 的层数
5. config=nf. Config(
6.       layers_per_proc=[2,2,2,2],
7. )
8.
9. # 使用深度学习框架 API 定义的 DNN
10. model=tf. keras. Sequential([
11.       tf. keras. layers. Flatten(),
12.       tf. keras. layers. Dense(128),
13.       tf. keras. layers. Dense(64),
14.       tf. keras. layers. Dense(64),
15.       tf. keras. layers. Dense(64),
16.       tf. keras. layers. Dense(64),
17.       tf. keras. layers. Dropout(0. 2),
18.       tf. keras. layers. Dense(10),
19. ])
20.
21. # 传入使用深度学习框架 API 定义的 DNN、优化器、损失函数
22. nf_model=nf. NFModel(
23.       model,
24.       config=config,
25.       optimizer=optimizer,
26.       loss=loss,
27. )
28.
29. # 训练模型
30. nf_model. fit(
31.       x_train,y_train,
32.       epochs=epochs,
33.       batch_size=batch_size,
34. )
```

框架将会根据用户设定的配置参数值自动将模型拆分到不同的计算节点上并微调模型结构，生成此模型的并行训练版本。在模型并行中，模型分区之间的通信通过 MPI 通信库中的 send()和 recv()实现。

3.3 用户透明的自动 DNN 拆分

传统流水线并行算法要求用户手动拆分、改写模型代码，并将模型参数手动分配给各个计算节点，具有一定的技术难度。为了实现用户透明的模型并行，本文设计实现了一种自动将 DNN 模型以“层”为基本单位拆分成多个独立分区的模型并行方法。

本框架仅需用户提供简单的 DNN 模型拆分需求（即每个模型分区有几层），即可按照原 DNN 模型层的顺序依次自动选取满足要求的层，并在合适的位置拆分原模型。如图 1 所示，对于每一次拆分操作，被分开的 DNN 模型将分别形成两个独立的模型分区：处于数据流上游的 D1 模型分区被称为上游模型；而处于数据流下游的 D2 模型分区被称为下游模型。在此拆分过程中，上游模型的最后一层将成为上游模型的输出层，为了使下游模型能够接收数据，将在每个下游模型前添加一个特殊的“梯度层”（图 1 中用 Grad 表示）作为下游模型新的输入层。这一自动化过程免去了用户手动将 DNN 模型拆分并添加数据传输层的麻烦，同时也保证了拆分后的 DNN 模型的有效性。拆分后每个模型分区将会由不同的计算节点负责存储和计算。前向传播时，上游模型的输出通过网络传输，传递到另一个计算节点中的下游模型中作为输入；反向传播时，每个模型分区（第一个分区除外）中的梯度层负责将梯度信息通过网络传递给上游的模型分区。

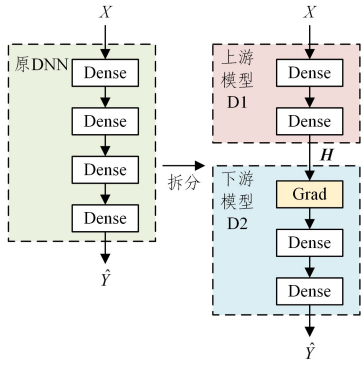


图 1 自动 DNN 模型拆分

Fig. 1 Automatic DNN splitting

3.3.1 自动 DNN 模型拆分的条件：有向无环图

DNN 模型一般由若干层组成，可以将一个 DNN 视为一张图，其中每个层（或某几个层的组合）都是图中的一个节点。当层 B 的输入依赖于层 A 的输出时，可以通过一条由层 A 到层 B 的有向边来表示，这个图被称为“层依赖”图。为了确保模型并行的可行性，需要保证被切分的 DNN 模型的层依赖

图是一个有向无环图。这也是使用本框架进行流水线并行的必要条件，否则，循环依赖的层之间会互相等待，形成死锁，阻止流水线并行的正常运行。

图 2 是一个由 5 个神经网络层组成的 DNN 模型，层 A 是输入层，层 E 是输出层。其中层 A、层 BCD 的组合、层 E 三者构成一个有向无环图。可按虚线拆分为 3 个模型分区，此时层 B、层 C、层 D 所在的节点有唯一的数据流 A→(BCD)→E。由于层 B、层 C、层 D 三者构成了有向有环图，因此它们不可拆分，即层 B、层 C、层 D 必须同时被放置于同一个计算节点上。假设层 B 与层 CD 不在同一节点上，则承载 B 的计算节点将会有两个不同的数据流：A→B→(CD)和(CD)→B→E。就如工厂流水线中的工人无法在一个流水线环节上同时处理两件事一样，在流水线并行训练中，这一计算节点也无法同时处理两个数据流，导致死锁，无法正常训练。

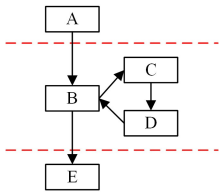


图 2 由 5 个网络层组成的层依赖图

Fig. 2 Layer dependency graph containing five layers

3.3.2 自动 DNN 模型拆分的关键：梯度层

每一次模型拆分均会将模型分为上、下游模型两个部分，然而用户定义的损失函数只有一个，直接用于下游模型中。因此，重新定义一个用于上游模型的损失函数成为实现模型拆分后的反向传播的关键。为了在模型分区直接有效传播梯度信息的同时最小限度地修改模型结构(该修改无需人工进行)，本框架采用在下游模型的输入端自动添加“梯度层”作为解决方案。不失一般性地，我们设整个 DNN 被拆分为如图 1 中 D1 和 D2 两部分。整个网络输入为 $\boldsymbol{X}$，模型输出为 $\hat{\boldsymbol{Y}}$，训练目标为 $\boldsymbol{Y}$。模型拆分后，每个分区可依次抽象为 D1：$\boldsymbol{H}=f_\theta(\boldsymbol{X})$；D2：$\hat{\boldsymbol{Y}}=g_\phi(\boldsymbol{H})$($\theta$，$\phi$ 为各分区的可训练参数)。则该 DNN 可以表示为 $\hat{\boldsymbol{Y}}=g_\phi(f_\theta(\boldsymbol{X}))$，损失函数为 $l=L(g_\phi(f_\theta(\boldsymbol{X})),\boldsymbol{Y})$。这样根据链式法则，损失函数对于下游模型 D2 的参数 ϕ 的梯度为：

$$\frac{\partial l}{\partial \phi}=\frac{\partial l}{\partial \hat{\boldsymbol{Y}}}\cdot\frac{\partial \hat{\boldsymbol{Y}}}{\partial \phi}$$

得益于 l，$\hat{\boldsymbol{Y}}$ 和 ϕ 均存储在下游模型所在的计算节点，因此上式可以通过深度学习框架的自动微分机制很方便地求得。但是，想得到损失函数对于上游模型 D1 的参数 θ 的梯度则并不容易。

$$\frac{\partial l}{\partial \theta}=\frac{\partial l}{\partial \hat{\boldsymbol{Y}}}\cdot\frac{\partial \hat{\boldsymbol{Y}}}{\partial \boldsymbol{H}}\cdot\frac{\partial \boldsymbol{H}}{\partial \theta}$$

由于 $\hat{\boldsymbol{Y}}$ 只存在于 D2 分区所在节点中，而深度学习框架的自动微分机制没有提供一种求损失函数关于不在本地的变量的梯度的方式，我们无法直接在 D1 分区中计算上式。此时，“梯度层”就发挥了作用。梯度层在前向传播时将输入直接转发至下一层，不起任何作用；但在反向传播时则负责将损失函数对于输入的梯度传递至存放上游模型的节点，即 D1 分区可以直接从 D2 分区接收 $\frac{\partial l}{\partial \boldsymbol{H}}$。这样便可以在 D1 分区中方便地求出损失函数关于 θ 的梯度。

3.4 面向通用计算设备的流水线并行

模型并行的中心思想是对模型进行分区，不同的计算节点负责计算不同的模型分区，从而减轻大参数量 DNN 对单个计算节点造成的内存压力。但由于深度神经网络的计算具有有序性，简单将不同模型参数放到不同的计算节点中顺序计算会导致前向或反向传播期间只有一个(或一组)计算节点处于活跃状态，从而严重浪费计算能力。如图 3(a)所示，$F(x,y)$ 表示第 x 个批次中第 y 个微批次的前向传播过程，$B(x,y)$表示第 x 个批次中第 y 个微批次的反向传播过程，灰色表示空闲的时间片。空闲时间片占整体时间的比例较高，浪费了不少机器算力。

流水线并行算法是一种高效且适用范围广的模型并行方式，可以尽可能使同一时间处于活跃状态的计算节点增多，从而充分利用计算资源，提高模型训练效率。同时，它避免了向量并行中向量拆分与合并时容易出错和效率低下的问题[10]，也不存在 ZeRO 并行不受通用计算设备支持的问题[14]。因此，本框架采用流水线并行算法作为模型并行的具体实现策略。如图3(b)所示，流水线并行在对模型进行划分后，将每个训练批次划分为更小的微批次。微批次的划分原则如下：

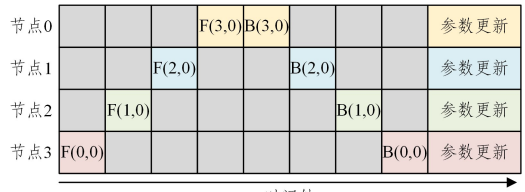
1)微批次的数量：一个训练批次最多可以被划分为不大于训练批次中数据条数个微批次，即需要保证每个微批次中至少有一条数据。从图 3(b)中可以看出，一个训练批次划分出的微批次的数量越多，各个计算节点并行计算的时间片占比越大，空闲时间片占比越小，系统资源利用越充分。

2)微批次中数据条数：不同微批次中数据条数应该大致相等，从而减小计算设备利用率波动。单个微批次中的数据最大条数受到计算设备承载能力的限制。

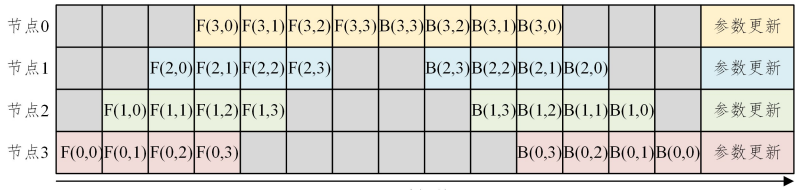
3)训练效果与训练效率的平衡：虽然增大微批次数量可以减小空闲时间片占比，增加微批次中数据条数可以最大化利用计算设备的承载能力，增加数据吞吐量。但是，实际训练批次大小等于微批次数量乘以单个微批次数据条数(假设不同微批次的数据条数相等)。而过大的训练批次大小可能会使 DNN 模型无法正常训练。因此，用户需要结合实际需求合理选取微批次的数量和微批次中的数据条数。

在图 3(b)中，一个批次被划分成了 4 个微批次。通过在微批次这一级别采取流水线式的操作，多个计算节点间的微批次可以并行计算，大大提高了整体计算效率。每个微批次所对应的梯度将会在同一个批次中不断累积，以保持与非模型并行训练效果的一致性。

GPipe[12]等现有的流水线并行实现往往与固有框架深度整合，无法适用于国产加速设备等通用计算设备。本框架基于主流深度学习框架(TensorFlow，PyTorch 等)和 MPI 通信协议实现了一种高效的流水线并行算法，开发了流水线并行的 MPI 通信域划分方式和通信方式，在保证计算正确性的同时也进行了效率优化。



(a)原始模型并行策略



(b)流水线并行策略

图 3 原始的模型并行与流水线并行策略

Fig. 3 Naïve model parallel strategy and pipeline parallel strategy

3.4.1 前向传播

在流水线并行中，前向传播需要每个节点完成获取数据、处理数据、发送数据 3 个步骤。第一个节点直接从训练数据集中获取数据，最后一个节点需要把处理后的数据发送给损失函数。由于需要保证每个节点在成功发送数据后立即获取下一个数据，这一过程涉及分布式计算的节点同步问题。因此，本框架巧妙地利用了 MPI 中 send()和 recv()方法会自动阻塞的特性进行同步。通过这一方式，在使用高效的 MPI 通信协议的同时，保证了节点之间计算的正确同步。前向传播的伪代码如算法 1 所示(以 TensorFlow 为例)。

算法 1 前向传播

输入：模型(model)，微批次(microbatches)

输出：预测值(predictions)，梯度记录(tapes)，损失值(losses)

1. 初始化空列表：predictions，tapes，losses
2. 对于每一个微批次 microbatch，执行第 3－9 步：
3. 将 microbatch 分解为输入数据和真实标签 y_true
4. 使用 tf. GradientTape 记录梯度
5. 如果当前节点是第一个节点：
   5.1. 通过模型 model 计算预测值 y_pred
   5.2. 将 y_true 发送到最后一个节点
   5.3. 将 y_pred 发送到下一个节点
6. 如果当前节点是最后一个节点：
   6.1. 从第一个节点接收 y_true
   6.2. 从前一个节点接收中间结果 recvd
   6.3. 通过模型 model 计算最终的预测值 y_pred
   6.4. 计算损失值并将其添加到 losses 列表中
7. 如果当前节点既不是第一个节点也不是最后一个节点：
   7.1. 从前一个节点接收中间结果 recvd
   7.2. 通过模型 model 计算预测值 y_pred
   7.3. 将 y_pred 发送到下一个节点
8. 将 y_pred 添加到 predictions 列表中
9. 将梯度记录添加到 tapes 列表中
10. 返回 predictions，tapes，losses

3.4.2 反向传播

反向传播时，自动添加进模型中的梯度层与 TensorFlow 中 GradientTape 类中的 gradient()方法结合，或与 PyTorch 中 autograd 模块中的 grad()方法结合，获取该模型分区的求导中间结果。具体来讲，当对每一个加入了梯度层的模型分区使用“tf. GradientTape. gradient()”或“torch. autograd. grad()”时，该方法不仅会返回模型分区中各个参数对应的梯度，还会返回中间求导结果。梯度用于稍后更新该层的参数，中间求导结果则会被发送至序号较小的节点(上游模型)。在反向传播中，本文也利用了 MPI 中 send()和 recv()方法会自动阻塞的特性进行同步。通过这一方式，既保留了自动微分机制的优点，也增强了模型与流行深度学习框架代码的兼容性。反向传播的伪代码如算法 2 所示(以 TensorFlow 为例)。

算法 2 反向传播

输入：预测值(predictions)，梯度记录(tapes)，损失值(losses)、模型(model)，微批次数量(n)

输出：更新后的模型(new_model)

1. 初始化空列表：gradients
2. 对于每一个微批次 i(i 从 0 取到 n－1)，执行第 3－6 步：
3. 如果当前节点是第一个节点：
   3.1. 从下一个节点接收误差 error
   3.2. 执行 TensorFlow 反向传播算法：结合 error，使用 tapes[i]计算 predictions[i]对于模型 model 的梯度 gradient
4. 如果当前节点是最后一个节点：
   4.1. 执行 TensorFlow 反向传播算法：使用 tapes[i]计算 predictions[i]对于模型 model 的梯度 gradient
   4.2. 从梯度层获取误差 error
   4.3. 将 error 发送到前一个节点
5. 如果当前节点既不是第一个节点也不是最后一个节点：
   5.1. 从下一个节点接收误差 error
   5.2. 执行 TensorFlow 反向传播算法：结合 error，使用 tapes[i]计算 predictions[i]对于模型 model 的梯度 gradient
   5.3. 从梯度层获取误差 error
   5.4. 将 error 发送到前一个节点
6. 将计算得到的 gradient 添加到 gradients 列表中
7. 计算所有 gradients 的和
8. 使用优化器应用计算得到的梯度来更新模型 model，得到更新后的模型 new_model
9. 返回 new_model

3.4.3 可配置的层重用

流水线并行训练中存在层重用问题。传统的流水线并行把模型拆分成多个子网络，按顺序放到不同的计算节点上执行。但有些模型中的某些层，如 Transformer 语言模型中的词嵌入层，需要在模型的不同部分(即流水线并行中的不同计算节点)重复使用。由于需要被重用的层不存在于所有计算节点中，纯粹的流水线并行就无法实现，需要引入层重用技术。

本框架实现了一种层重用机制。它在模型定义中增加了一个参数，用于标记并区分不同的重用层。训练开始时，重用层的每个计算节点会复制这个重用层的参数到本地。前向传播时，各个重用层副本会在不同的计算节点上正常参与计算。在所有反向传播结束后，对所有重用层副本的梯度执行额外的全规约操作。这个规约操作可以保证重用层在不同流水线阶段中的权重同步。这样，流水线并行训练就可以灵活实现层的重用，扩大并行范围，使之能够应用到更广泛的模型中。

3.5 分布式训练整体流程

整个分布式训练框架的流程如图 4 所示。

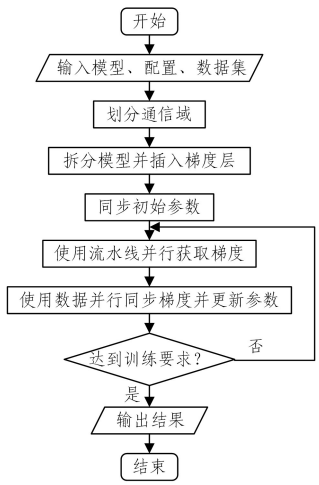


图 4 分布式训练流程图

Fig. 4 Flow chart of distributed training

首先，输入模型、配置以及数据集，为后续的并行训练做准备。随后，框架将模型自动拆分为多个子模型并划分 MPI 通信域。每个子模型中均被插入一个梯度层，这有助于后续的并行计算和参数同步。在此之后，框架在不同节点间同步模型初始参数，确保所有子模型的初始参数保持一致。接下来，训练过程进入“使用流水线并行获取梯度”的阶段。在这一阶段，各个子模型并行地计算梯度，并使用流水线方式提高计算效率，从而加速训练过程。在实际的 DNN 模型分布式训练场景中较少单独使用模型并行训练，通常会结合数据并行一起提升大模型训练效率。因此，本框架也被设计成可支持多种数据并行算法的框架。在流水线并行计算一轮后，框架使用数据并行算法将各个数据并行组中的子模型计算得到的梯度进行同步并更新参数。在每次参数更新后，框架会判断是否达到了训练要求。如果训练要求尚未满足，则返回到“使用流水线并行获取梯度”的阶段，继续进行下一轮的并行训练；一旦训练要求满足，框架就将训练得到的模型或结果输出。这样的分布式训练框架能够充分利用多计算节点的优势，加速模型训练过程，提高训练在时间、空间上的效率。

4 实验结果与分析

为验证本文实现的模型并行算法的效率与正确性，我们设计了完整、详尽的测试实验。实验覆盖不同类型的计算设备、不同规模的模型和不同规模的集群的节点数量，以充分验证算法的可行性和扩展性。

在实际的分布式并行训练中较少单独使用模型并行，通常会结合数据并行一起支撑大模型的训练。在本实验中，为了获得更加接近实际使用场景的测试，本文的模型并行也将和数据并行结合使用。

实验中使用两种规模的模型验证本框架在模型参数上的可扩展性，两个模型分别为小规模的图像分类卷积神经网络和超大规模自然语言预训练 GPT 模型。其次，本实验使用不同规模的计算节点验证本框架在节点数量上的可扩展性。实验还分别在 CPU，GPU，DSP 这 3 种计算设备上验证了本框架的正确性。

4.1 计算设备通用性验证

本实验选用了 CPU，DSP，GPU 这 3 种硬件实验平台。其中，CPU 和 DSP 实验硬件环境为天河新一代超算系统，其拥有大规模的 MT-3000 异构计算节点[6]，并采用高速网络互联结构[20]。每个 MT-3000 节点由 1 个 CPU 和 4 个 DSP 簇组成。

GPU 实验硬件环境为 NVIDIA Tesla V100S 32 GB。3 种硬件平台的软件环境均为 Ubuntu 20.04.2 LTS，PyTorch 1.10.0，TensorFlow 2.12.0。

在 CPU，DSP，GPU 环境下，本框架均可正常接收用户输入，自动拆分模型，执行流水线并行计算，输出结果。这证明本框架可适配多种计算设备，具有较好的通用性。

4.2 卷积神经网络验证

本节采用卷积神经网络模型验证本框架的正确性。任务目标是识别 MNIST 数据集[21]中的手写数字图片。所使用的深度学习模型依次包含：输入层，两个卷积核为 3×3、通道数分别为 32，64 的二维卷积层，2×2 池化层，概率为 0.25 的暂退层，输出维度为 128、激活函数为 ReLU 的全连接层，概率为 0.5 的暂退层，输出维度为 10、激活函数为 softmax 的全连接层。

实验分别将完整模型拆分成 2 段或 3 段模型分区进行流水线并行，同时辅以 1～6 组数据并行进行测试，共训练 5 个周期，批次大小为 1000，并均匀拆分为 10 个微批次，每个微批次中包含 100 条训练数据。实验结果如表 1 所列。其中，当数据并行组数为 1 时，无数据并行，仅模型并行。流水线分区数为 1 分区时，模型没有进行拆分，不进行模型并行。数据并行组数为 1 且流水线分区数为 1 时等价于不采用分布式并行方式的单节点模型训练。从表中可以看出，无论是模型并行还是数据并行，对模型训练都有一定的加速效果(使用模型并行时会增加参与计算的计算节点)。随着数据并行组数的增加，加速效果也变得更加明显。在数据并行组数相同时，模型被拆分得越细，训练的效率也略有降低，这是因为模型并行会存在模型分区间通信的开销。

表 1 卷积模型并行训练时间

Table 1 Parallel training time of CNN model

(s)

数据并行组数	流水线分区数		
	1 分区	2 分区	3 分区
1	670.25	608.08	589.76
2	411.93	338.44	334.52
3	241.91	234.05	231.09
4	183.48	184.11	186.40
5	150.08	157.52	156.86
6	125.93	135.25	138.91

相较于不进行任何并行，混合并行的训练加速比如表 2 所列。从表中可以看出，随着节点数量的增加，加速性能也有着不错的表现，验证了本框架在节点数上的可扩展性和对数据并行的较好的兼容。在不使用模型并行时，并行效率（加速比除以数据并行组数）基本不变；在使用流水线并行时，由于会增加额外的模型内部通信开销，因此加速比会低于仅用数据并行时的情况，属于正常现象。

表 2 卷积模型并行训练加速比

Table 2 Parallel training speedup ratio of CNN model

数据并行组数	流水线分区数		
	1 分区	2 分区	3 分区
1	1.00	1.10	1.13
2	1.62	1.98	2.00
3	2.77	2.86	2.90
4	3.65	3.64	3.59
5	4.46	4.25	4.27
6	5.32	4.95	4.82

训练时，所有模型的损失函数正常下降，所有训练得到的模型在 MNIST 测试集上均能达到不低于 97%的准确率。总体来看，本文的流水线并行框架在深度学习模型分布式训练上能够取得较优的效果。

4.3 大型语言模型验证

本文通过训练 GPT 模型的实验来验证本文流水线并行框架对不同规模模型的适应性以及对大规模模型训练的支持。GPT 模型参数如表 3 所列。模型具体的参数量由堆叠的解码器个数决定。

表 3 GPT 模型详细参数

Table 3 Details of GPT model used in experiment

参数名	参数值
词表大小	13312
词嵌入维度	1024
隐含维度	1024
自注意力头数	16
自注意力步长	128
自注意力表达性	8
窗口大小	1024

实验成功验证了表 4 所列的混合并行配置组合的效果。可以看到，随着计算节点数量的增加，可支持的模型参数也越来越大。为了确保实验的效率，同时将模型拆分成更多的模型分区，本实验在单个计算节点上（即单个模型分区内部）仅存放少量的模型参数，最终使用 2 500 个模型分区成功容纳了一个千亿参数大模型。通过增大单个模型分区的参数量，还可在减小模型分区的数量的同时保持模型整体参数量不变。这证明本框架在模型参数量、模型分区数量和计算节点数量上的可扩展性。本框架已经可以利用一万个以上的计算节点，能够支撑起千亿级参数大模型的正确训练。

表 4 GPT 模型训练配置

Table 4 GPT model training configuration

数据并行组数	流水线分区数	使用计算节点数	等效模型参数量	全局批次大小
1	1	1	约 40 M	4
1	4	4	约 160 M	4
4	1	4	约 40 M	16
4	25	100	约 1 B	16
4	125	500	约 5 B	16
4	250	1000	约 10 B	16
4	2500	10000	约 100 B	16

结束语 本文提出并实现了一种面向通用计算设备的自动流水线并行分布式训练框架。在天河新一代超算系统上的实验验证了多种不同规模模型的自动拆分和流水线并行计算，尤其是利用大规模计算节点验证了本文的自动流水线并行框架对大模型训练的有效支撑。测试了覆盖不同模型尺寸、不同节点规模、不同计算设备，全面验证了所提框架的正确性与有效性，为通用计算设备上的大规模分布式深度学习提供了关键技术支持。但实现过程中，仍有部分技术细节可以进一步完善和优化，例如，自动平衡内存负载，在大数据进行传输前进行压缩，根据路径负载自动划分模型等。这些工作将是本文工作的后续优化方向。

参 考 文 献

[1] BROWN T B，MANN B，RYDER N，et al. Language models are fewshot learners[C]// Proceedings of the 34th International Conference on Neural Information Processing Systems. 2020：1877-1901.

[2] FEDUS W，ZOPH B，SHAZEER N. Switch transformers：Scaling to trillion parameter models with simple and efficient sparsity[J]. arXiv：2101.03961，2021.

[3] HE K，ZHANG X，REN S，et al. Deep residual learning for image recognition[C]// 2016 IEEE Conference on Computer Vision and Pattern Recognition. 2016：770-778.

[4] HE K，ZHANG X，REN S，et al. Identity mappings in deep residual networks[C]// Computer Vision-ECCV 2016. Springer，2016：630-645.

[5] NVIDIA. CUDA toolkit[EB/OL]. https://developer.nvidia.com/cuda-toolkit.

[6] LU K，WANG Y，GUO Y，et al. MT-3000：a heterogeneous multi-zone processor for HPC[J]. CCF Transactions on High Performance Computing，2022，4(2)：150-164.

[7] AWAN A A，CHU C，SUBRAMONI H，et al. OC-DNN：exploiting advanced unified memory capabilities in CUDA 9 and volta gpus for out-of-core DNN training[C]// 25th IEEE International Conference on High Performance Computing. IEEE，2018：143-152.

[8] MARKTHUB P，BELVIRANLI M E，LEE S，et al. DRAGON：breaking GPU memory capacity limits with direct NVM access[C]// Proceedings of the International Conference for High Per-

formance Computing, Networking, Storage, and Analysis. IEEE/ACM, 2018: 32: 1-32: 13.

[9] RHU M, GIMELSHEIN N, CLEMONS J, et al. vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design[C]// 49th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, 2016: 18: 1-18: 13.

[10] SHOEYBI M, PATWARY M, PURI R, et al. Megatron-lm: Training multi-billion parameter language models using model parallelism[J]. arXiv: 1909. 08053, 2019.

[11] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[C]// Proceedings of the 31st International Conference on Neural Information Processing Systems. 2017: 6000-6010.

[12] HUANG Y, CHENG Y, BAPNA A, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism[C]// Proceedings of the 33rd International Conference on Neural Information Processing Systems. 2019: 103-112.

[13] RASLEY J, RAJBHANDARI S, RUWASE O, et al. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters[C]// Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. ACM, 2020: 3505-3506.

[14] RAJBHANDARI S, RASLEY J, RUWASE O, et al. Zero: memory optimizations toward training trillion parameter models [C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE/ACM, 2020.

[15] REN J, RAJBHANDARI S, AMINABADI R Y, et al. Zero-offload: Democratizing billion-scale model training[C]// 2021 USENIX Annual Technical Conference. USENIX Association, 2021: 551-564.

[16] RAJBHANDARI S, RUWASE O, RASLEY J, et al. Zero-infinity: breaking the GPU memory wall for extreme scale deep learning[C]// International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2021.

[17] ZHAO Y, GU A, VARMA R, et al. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel[J]. Proceedings of the VLDB Endowment, 2023, 16(12): 3848-3860.

[18] BI R, XU T, XU M, et al. PaddlePaddle: A Production-Oriented Deep Learning Platform Facilitating the Competency of Enterprises[C]// 24th IEEE International Conference on High Performance Computing & Communications; 8th International Conference on Data Science & Systems; 20th Int Conf on Smart City; 8th International Conference on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys). IEEE, 2022: 92-99.

[19] KIM T, KIM H, YU G, et al. BPipe: Memory-Balanced Pipeline Parallelism for Training Large Language Models[C]// Proceedings of Machine Learning Research: International Conference on Machine Learning. PMLR, 2023: 16639-16653.

[20] GONG C, LIU J, BAO W, et al. Review on Ecological Construction of Domestic High-performance Parallel Application Software in Post Moore Era[J]. Journal of System Simulation, 2022, 34(10): 2107-2118.

[21] DENG L. The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web][J]. IEEE Signal Processing Magazine, 2012, 29(6): 141-142.

ZHONG Zhenyu, born in 1997, Ph. D candidate. His main research interests include natural language processing, high performance computing and AIOps.

SUN Yufei, born in 1976, Ph. D, professor. Her main research interests include deep learning, heterogeneous computing, artificial intelligence, etc.

(责任编辑: 何杨)