

订阅DeepL Pro以编辑此演示文稿。  
访问[www.DeepL.com/pro](https://www.deepl.com/pro?cta=edit-document)，了解更多信息。

|  |
| --- |
| 2022 |

Colossal-AI：用于大规模并行训练的统一深度学习系统

李胜贵1 , 方家瑞1 , 卞正达1 , 刘洪新1 , 刘玉良1 , 黄海晨1 , 王伯祥1 , 游杨1,2

flisg, fangjr, bian.zhengda, liuhongxin, liuyuliang, hhc, boxiangwg@hpcaitech.com

youy@comp.nus.edu.sg

1HPC-AI技术公司

* 新加坡国立大学（NUS）

杨友是新加坡国立大学的一名教师，这项工作是在HPC-AI科技公司完成的。

|  |
| --- |
| arXiv:2110.14883v2 [cs.LG] 20 Sep |

摘要--Transformer模型的成功将深度学习模型的规模推向了数十亿的参数。由于单个GPU的内存资源有限，然而，选择最佳并行策略的最佳实践仍然缺乏，因为它需要深度学习和并行计算方面的专业知识。

Colossal-AI系统通过引入一个统一的接口，将模型训练的顺序代码扩展到分布式环境中来解决上述挑战。它支持并行训练方法，如数据、管道、张量和序列并行，以及与零冗余优化器相互结合的异质训练方法。与基线系统相比，Colossal-AI在大规模模型上的训练速度可以达到2.76倍。

Index Terms-Machine Learning, Deep Learning, Distributed System, Parallel Computing, Heterogeneous System

1. 简介

深度学习已经在许多应用中展示了其令人印象深刻的性能，并在传统方法难以解决的问题上带来了突破。在大量数据的情况下，神经网络，如BERT[[1]](#page12)和Vision Transformer[[2]，](#page12)能够学习高维特征并进行预测，其水平甚至是人类无法匹敌的。随着强大硬件的出现，神经网络的架构更加多样化，参数数量也更多。

人工智能社区的一个明显趋势是，深度学习模型正在迅速变大。仅仅用了三个月的时间，最大模型的头衔就从BERT-Large转移到了2020年的GPT-2[[3]](#page12)，而GPT-2的参数数量是BERT-Large的五倍左右。此外，GPT-2进一步发展为GPT-3[[4]](#page12)，拥有1750亿个参数。最近，GLM[[5]](#page12)以令人惊讶的1.75万亿个参数夺得冠军。这些大规模的模型消耗了更多的数据和计算资源，通常在许多领域比它们较小的同类模型具有更好的通用性和性能。随着更强大的计算硬件和更大的数据集进入人们的视野，这种趋势有望继续下去。因此，传统的训练方法已不再有效，分布式训练成为实现大规模模型训练的必要条件。

常用的加速器硬件（如GPU）的有限快速内存是将模型扩展到数十亿个参数的瓶颈。由于自适应优化器[[6], [7]](#page12)被广泛应用于巨型模型训练中，深度学习的内存消耗来自模型参数、层激活、梯度和优化器状态。我们把模型参数、梯度和优化器状态称为模型数据，把层激活称为非模型数据。模型数据的总内存消耗可能比单纯的参数消耗大几倍，这使得单个GPU不再能满足大规模模型训练的需要。100亿个Float 16格式的参数已经可以消耗20GB的模型内存，而一个典型的GPU只有16或32GB的内存。在没有任何优化的情况下，用一个数据样本训练一个100亿个参数的模型可能要耗费超过80GB的内存，这远远超过了一般GPU的内存。

数据并行是深度学习框架所支持的最流行的并行方法，因为它易于实现。尽管它很受欢迎，但这种方法需要在单个设备上持有完整的模型，这使得它只适合于小规模的模型，如ResNet[[8]。](#page12)为了减少GPU上的内存占用，人们提出了许多方法。激活检查点[[9]](#page12)被用来通过用计算换取内存来减少非模型数据。为了解决模型数据的内存限制，人们探索了诸如流水线并行[[10]、[11]](#page12)和10-sor并行[[12]](#page12)等并行化技术来分拣模型数据并提高训练效率，使之有可能在更大范围内训练模型。目前最先进的系统为扩展挑战提供了解决方案，包括GShard[[13]，](#page12)FairScale[[14]，](#page12)以及Megatron-LM[[15]](#page12)和DeepSpeed[[16]。](#page12)在这些系统中，Megatron-LM和DeepSpeed是开源社区中最受欢迎的，并且提供了最好的性能。因此，它们被选为我们实验的基线。Megatron-LM被提议通过利用OPT-mized管道和张量并行性来训练基于Transformer的模型。同时，DeepSpeed团队的研究表明，数据并行在内存冗余方面有一个内在的缺陷。他们提出了一种有效的方法来划分模型相关的数据，以充分

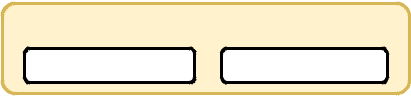
消除数据并行训练中的内存冗余。这两种高效的方法为将模型训练扩展到数百个设备和数十亿个参数铺平了道路。

由于大多数深度学习工程师和研究人员都习惯于编写非分布式代码，因此他们要适应并行和分布式编程是有一定难度的。然而，目前还没有一个系统可以让他们轻松地使用所有的加速方法。Megatron-LM的一个关键问题是，它从多设备通信、数据一致性和扩展效率方面引入了额外的工程努力。如果我们想探索更多维度的并行性，就更是如此。同时，还有其他方面需要注意，如环境设置和随机性。这样的技能障碍并不能帮助一般的人工智能社区应对日益增长的多维分布式训练的需求。至于DeepSpeed，尽管它有新颖的零冗余优化器，但它只提供了数据并行训练的优化，并且严重依赖与Megatron-LM的集成来实现混合并行训练。其复杂的实现方式也导致退化的

**培训师**

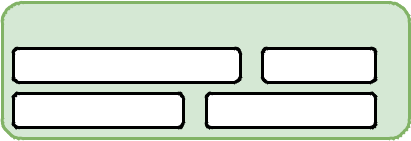


钩子



**执行引擎**

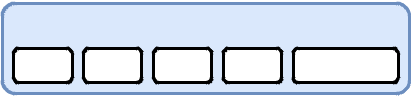
时间表 钩子



**加速**

自动混合精度 卸载

模型分片 优化器分片



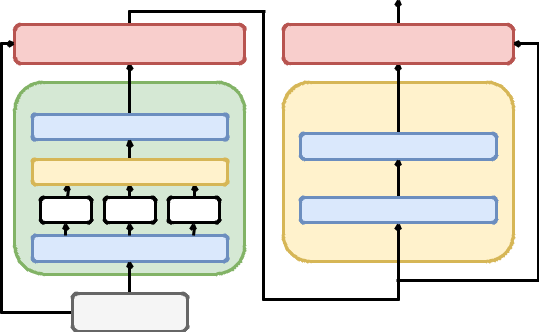
**分布式操作系统**

1D 2D 2.5D 3D 序列



**平行上下文管理器**

图1：Colossal-AI的架构



系统性能和可扩展性差。

为了解决这些问题，我们开发了Colossal-AI，这是一个开源系统，通过在一个深度学习系统中统一一系列的训练加速技术，使人工智能社区的复杂分布式训练民主化。在这个系统中，我们还包括了新颖的并行方法，如多维张量并行和序列并行。Colossal-AI旨在通过提供用户友好的API使分布式训练变得简单，同时允许用户保持他们编写单节点的编码习惯。

**添加和归一化**

**多头关注**

**投影（线性）**

**缩放点积注意**

**查询** **钥匙** **价值**

**QKV\_投影 (线性)**

**嵌入数据**

**添加和归一化**

**馈送**

**投影（线性）**

**投影（线性）**

方案。简而言之，我们在这项工作中为大规模分布式培训带来了以下主要贡献：

1 Colossal-AI是一个统一的深度学习系统，为人工智能社区提供了最完整的加速技术，如数据、管道。如图[1](#page2)所示的模块化设计[，](#page2)Colossal-AI允许自由组合这些技术以达到最佳的训练速度。同时，在设计模块时也考虑到了可扩展性，Colossal-AI可以轻松支持用户定制的功能。系统结构的细节将在实施部分讨论。

* Colossal-AI中提供了优化的并行性和异质训练方法。这些方法实现了比基线系统更好的系统性能。它们通过友好的API提供给用户，并带有

最低限度的代码修改。

3 我们进行了深入的分析，以研究不同硬件条件下适合的并行策略。

1. 背景情况

由于Transformer架构的出现，深度学习模型在计算机视觉和自然语言处理等领域获得了空前的性能提升。BERT[[1]](#page12)和Vision Transformer[[2] 。](#page12)

图2：变压器层的结构

已经成为他们各自领域的基本基础模型。图[2](#page2)显示了单个变换器层的典型结构[。](#page2)变换器层由一个多头注意块和一个前馈块组成。在一般的变形器层中，总共有四个线性层，由蓝色矩形代表。有了转化器层，我们可以沿着超参数扩展模型的大小，如层数、线性层的隐藏大小和多头注意模块中的注意头数。因此，模型的大小可以很容易地达到数十亿的参数。模型规模的纯粹增加带来了令人印象深刻的性能改善。例如，GPT-3（1750亿个参数）[[4]](#page12)在LAMBADA语言任务[[17]](#page12)上的预测精度绝对值增加了18%，超过了最先进模型的表现。

随着Transformer架构允许轻松扩展模型规模，计算和通信硬件变得更加强大，人工智能工程师和研究人员已经将训练范式从单GPU训练转向分布式训练，以追求更低的时间成本。分布式训练允许深度学习社区用更大的数据集来训练更大的模型，同时仍然保持可接受的工作完成时间。各种技术被

提出了加速分布式培训的建议，下面将讨论这些建议。

A.数据并行化

数据并行是最常见的并行形式，因为它很简单。在数据并行训练中，模型被复制到各个设备上，数据集被分割成若干碎片。如图[3a](#page4)所示，每个数据集碎片被分配到一个设备上，并送入相应设备上的模型[。](#page4)由于每个设备都有完整的参数集，有必要在每次训练迭代中同步模型权重。因此，在向后传播过程中计算参数梯度时，梯度将被降低，优化器将用同步的梯度更新参数。数据并行使训练和部署模型变得容易，并且由于额外的通信开销，与GPU的数量有亚线性的扩展。

B.模型平行性

在数据并行训练中，一个突出的特点是每个GPU持有整个模型权重的副本。然而，这使得大规模模型的内存要求很难满足，特别是在使用Adam[[18]](#page12)等有状态优化器时[。](#page12)当模型的规模增加时，GPU的内存成为瓶颈。因此，有必要对模型参数进行分片，使每个设备只需要持有模型的部分权重。因此，有人提出了模型并行化来解决这个问题。模型并行一般有三个子类别：张量并行、管道并行和序列并行。管线并行是将模型按层分割，每个设备持有部分层。

1）张量平行化

张量并行将张量分割在一个去维数组上，需要一个分布式的矩阵-矩阵乘法算法进行算术计算，如图[3b](#page4)所示[。](#page4)Megatron-LM[[12]](#page12)提出了1D张量并行，它在行或列维度上分割线性层，用于Transformer架构[[19]。](#page12)

以Transformer层的多层感知器（MLP）模块为例，我们可以把MLP模块看作是Y=W W21 X的矩阵乘法[，](#page4)如图[4](#page4)所示[，](#page4)其中X是隐藏状态，W1 和W2 是线性层权重，Y是层输出。图[4](#page4)中的激活函数和归一化层被忽略了[。](#page4)在一个典型的Transformer模型中，隐藏状态被W1 投射到更高的维度，然后被W2 投射回原始维度。我们可以将矩阵W1 分到列维，矩阵W2 分到行维。这将导致每个设备上出现Y的部分结果。可以对部分结果进行全还原操作，以获得矩阵乘法的正确最终结果。变换器架构中的自我关注模块是以类似的分布式方式计算的。

这样，在N个设备上训练时，每个设备将持有1=N个MLP参数器。这使得模型的大小能够

随着设备数量的增加而增加，并且超出了单个设备的内存容量，即使设备之间需要额外的通信以确保算术的正确性。

目前的方法依赖于Megatron-LM[[12]](#page12)提出的一维张量并行制[。](#page12)然而，这种方法有一些缺点，损害了通信和内存效率。这个方法的主要问题之一是它依赖于集体通信的全还原操作。这使得它对那些在单个节点上的GPU之间没有高端全连接NVLinks的机器不友好。如图[5a](#page4)所示[，](#page4)有了完全连接的高性能GPU互连[，](#page4)任何一对GPU之间的通信带宽都会很高。然而，这样的高端硬件是昂贵和稀缺的。许多机器，甚至一些超级计算中心的机器，只有部分连接的GPU，如图[5b](#page4)所示[。](#page4)在这种GPU拓扑结构下，远距离设备之间的通信带宽比直接连接的GPU要低得多，因为通信必须通过PCIe总线。因此，低通信带宽会阻碍一维张量并行的全还原操作的效率。

此外，1D张量并行在层的输入和输出上有冗余的内存使用。以Transformer架构中的MLP层为例，如图[4](#page4)所示，MLP层的输入X和输出Y在不同的设备上是重复的[。](#page4)这种内存冗余限制了在有限的硬软件资源上可以训练的最大模型规模，对大规模分布式训练的民主化没有帮助。

除了一维张量并行，更高级的张量并行被引入到大规模模型训练中，即二维、2.5维和三维张量并行[[20]-[22]](#page12)，如图[6](#page5)所示。这些方法将传统的分布式矩阵乘法应用于深度学习训练，在内存和通信效率方面具有优势。因此，它们可以更好地应对不同的硬件规格。这为用户提供了一个选择，即使用最适合他们机器的张量并行技术。

二维张量并行[[20]](#page12)依赖于SUMMA和Can-non矩阵乘法算法[[23]-[25]](#page13)，并将输入数据、模型权重和层输出沿两个不同维度分割。给定N个设备，张量被分割成N个大块

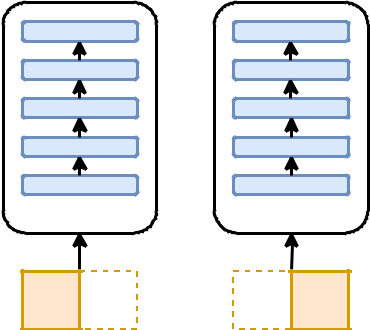
而每个块是由一个GPU拥有的。GPU被安排在p

在一个正方形的网络拓扑结构中，每一行有N个GPU p

并且有 N行，如图[6a](#page5)所示[。](#page5)例如，一个

形状为[P;Q]的张量将被划分为一个p p的大块张量

形状[P=N；Q=N]。这种方法导致了零内存冗余，因为输入和输出不再重复，与一维张量并行相比，导致内存消耗低得多。为了确保算术的正确性，部分输入和权重通过集体通信（如广播）在设备之间进行交流。尽管二维张量并行对模型权重有额外的通信开销，而一维张量并行则没有，但二维张量并行仍然更有效率，因为在二维张量并行中，由于更精细的分区方案，传输的数据量更少。

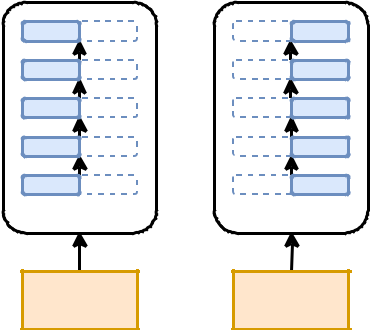


GPU 0 GPU 1

数据 数据

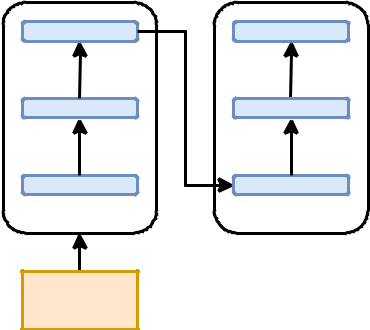
(a) 数据并行

|  |  |
| --- | --- |
| GPU 0 | GPU 1 |
| 数据 | 数据 |



(b) 张量平行

GPU 0 GPU 1



数据

(c) 管线并行

图3：现有的分布式训练的并行性

**GPU 0**

**GPU 1**

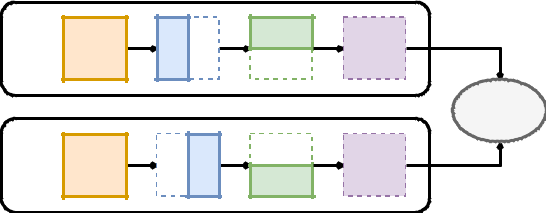
**X**

**X**

**W1**

**W1**

**W2**



**W2**

**部分**

**Y**

**全减**

**部分**

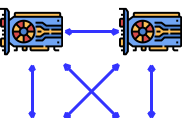
**Y**

三维张量并行[[22]](#page12)的提出是基于

三维矩阵乘法算法[[27]](#page13)，也集成到Colossal-AI中。如图[6d](#page5)所示，三维张量并行将张量以三维的方式分割[。](#page5)由于不是每个张量都有3个维度，我们只选择对第一和最后一个维度进行分割，其中第一维度将被分割为

两次。例如，一个形状为[P；Q]的张量将被划分为p3 2 p3

图4：Megatron-LM MLP模块



**视觉效果**



1. 完全连接的GPU



**视觉中国** **视觉中国**



**PCIe**



(b) 部分连接的GPU

图5：GPU计算节点上的常见网络拓扑结构

可扩展到大量的设备。

2.5D Tesnor并行[[21]](#page12)受到2.5D矩阵乘法算法[[26]](#page13)的启发，提出了进一步并行化二维张量并行的方法。它增加了矩阵的可选深度维度来实现并行化。当深度=1时，它接近于二维张量并行化。当深度>1时，它对矩阵进行3次分割，并增加了一个并行化程度。给定N个设备，张量的分割方式为：N = S2 D，其中S是正方形的一条边的大小，D是立方体的深度，如图[6b](#page5)所示[。](#page5)如果D是由用户给出的，S可以计算出来。当扩展到大量设备时，2.5D张量并行将比2D张量并行更有效，因为它进一步减少了通信量。

成一个形状为[P=N；Q=N]的块状张量。这种方法在N个设备上实现了最佳的O(N1=3 )通信开销，同时，通过优化的参数以及激活的负载平衡，计算和内存的使用都是均匀分布的。

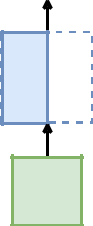
由于高级张量并行方法需要不同的网络拓扑结构，用户需要根据GPU的数量来选择方法。一维方法可以与任何数量的GPU一起工作，而二维、2.5维和三维方法分别需要n2 、a n2 和n3 GPU，其中a和n是正整数。当GPU的数量不能满足要求时，用户可以退回到1D张量并行制。

这些先进的张量并行方法在扩展到更多设备时提供了更低的通信量[[23],](#page12) [[24], [26], [27]。](#page13)同时，这些方法与流水线并行有更好的兼容性，因为它消除了在一维张量并行中存在的流水线阶段之间的通信开销。在Megatron-LM提出的一维张量并行制中，层的激活是重复的。因此，为了节省跨节点的通信带宽，张量在传递到下一个流水线阶段之前被分割成小块，在到达目标流水线阶段之后，小块被收集起来，进行节点内通信，如图[7](#page5)所示[。](#page5)相反，在先进的张量并行方法中，要传递到下一阶段的张量已经是整个逻辑张量的一个子块。因此，不需要分割收集操作，也不会产生额外的通信成本。

2) 管道并行化

除了张量并行，另一种分片方案是按层分割模型。与张量并行相比，管道并行更加友好，因为它不需要改变一个层的计算流程，特别是对于

**输出**



**砝码**

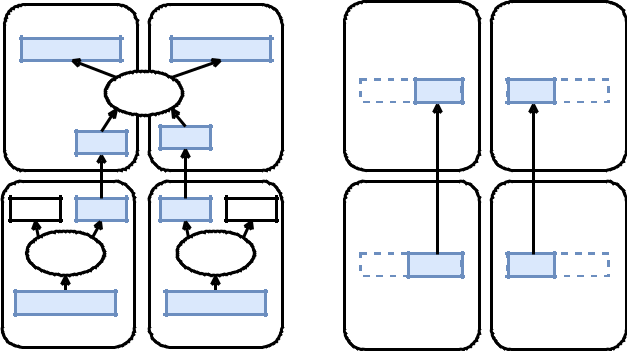


**输入**

(a) 一维张量并行 (b) 二维张量并行 (c) 2.5D张量平行主义 (d) 三维张量平行

图6：张量并行化

|  |  |  |  |
| --- | --- | --- | --- |
| **第一阶段** | **第一阶段** | **第一阶段** | **第一阶段** |
|  | **全体集合** | **激活** | **激活** |



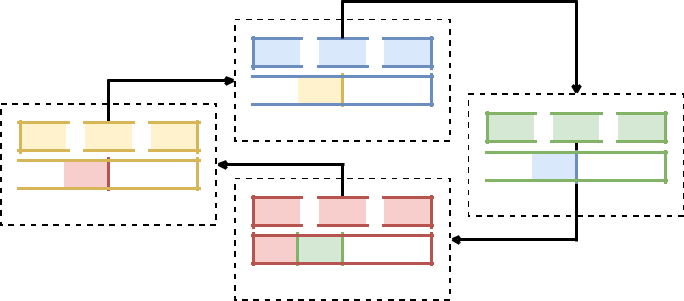
|  |  |  |  |
| --- | --- | --- | --- |
| **分割** | **分割** | **激活** | **激活** |
|  |  |
| **激活** | **激活** |  |  |
| **阶段0** | **阶段0** | **阶段0** | **阶段0** |
| (a) 一维张量并行 | | (b) 2//2.5/3D张量的平行性 | |

图7：管道阶段间的数据传输与张量并行性

基于动态图形的框架，如PyTorch[[28]。](#page13)诸如PipeDream[[29]](#page13)和GPipe[[30]](#page13)等方法被提出来，将模型分成若干块，每个块被分配给一个设备，如图[3c](#page4)所示[。](#page4)在前向传递过程中，每个设备将中间的激活传递给下一个阶段，并且

文件级别的文本理解。这是因为这些任务使用长序列长度的数据。由于Transformer层的自我注意机制相对于序列长度来说是二次复杂的，长序列的数据会增加中间激活所消耗的内存，限制了设备的训练能力。即使先进的张量并行方法可以对输入和激活进行分区，但模型参数带来的额外通信开销是没有必要的。

序列并行[[32]](#page13)被提出来，通过打破大序列维度带来的内存墙来实现长序列的建模。在序列并行中，模型像数据并行一样被复制到各个设备上。然而，输入数据不是按批次维度分割的，而是按序列维度分割的，每个设备只保留子序列。由于数据分布在各个设备上，需要额外的通信来完成自我注意的计算。在序列并行中，提出了环形自我关注模块来交换部分查询、密钥和值嵌入。



只计算最后一个流水线阶段的损失值。在后向传递过程中，每个设备将输入张量的梯度传回给前一个流水线阶段。每个流水线阶段的参数的梯度被累积起来，只有当流水线用完微批数据时，参数才被更新。管道平行性允许多个设备同时计算，导致更高的吞吐量。它还最大限度地减少了跨节点的通信，因为只有激活和梯度是在不同的节点之间进行通信的。

**查询 键 值**



**注意分数**



**设备3**

**查询 键 值**



**注意分数**



**设备0**

**查询 键 值**



**注意分数**

**设备2**

**查询 键 值**



**注意分数**



**设备1**

管线阶段。管道并行训练的一个缺点是会有一些冒泡时间，即当其他设备在进行计算时，一些设备是空闲的，导致计算资源的浪费。目前的方法如Chimera[[31]](#page13)使用双向流水线来减少冒泡时间，达到最先进的性能。

3) 序列平行化

张量并行主要解决的是模型数据带来的内存瓶颈。然而，非模型数据也可能成为AlphaFold 和Current等应用中的瓶颈。

图8：环形自我注意中的注意分数计算

自我注意机制的工作原理如下、

p

Attention(Q; K; V ) = softmax(QKT = dk )V

其中，Q是查询嵌入，K是密钥嵌入，V是值嵌入，dk 是密钥嵌入的维度。如同在序列并行中，查询、密钥和

值嵌入都是由序列二分法划分的，因此，我们需要以环形的方式传达部分嵌入来完成计算。以注意力分数的计算为例，注意力分数由A = softmax(QKT = dpk )得到。在环形自我关注模块中，密钥嵌入首先与本地设备上的查询嵌入相乘，然后转移到下一个设备上进行N 1次，如图[8](#page5)所示[。](#page5)通过这种方式，可以得到与本地子序列有关的部分注意力得分。最终的注意力输出AV可以用类似的方式计算。

序列并行是与管道并行正交的。与先进的张量并行类似，管道阶段之间不需要额外的通信，因为层的激活是由序列维度划分的。

C.零冗余数据并行化

上述方法主要解决模型参数数量带来的内存挑战。然而，模型参数只构成训练期间内存消耗的一部分。在模型数据中，优化器的状态会占到内存消耗的更大比例。当使用Adam等有状态的优化器时[[18]，](#page12)优化器的状态（即动量和方差）与模型参数所占的内存空间相比，可以占到三倍以上[[18]，](#page12)[[33]。](#page13)

在N个设备上的数据并行训练中，有N份模型参数、梯度和优化器状态。为了消除这种冗余，DeepSpeed[[16]](#page12)中提出了零冗余优化器，将这些冗余的模型数据分割到不同的设备上。零冗余优化器有三个阶段，逐个阶段将优化器状态、梯度和模型参数加入到分区方案中。以最后一个阶段为例，每个设备只持有1=N的模型数据，只更新相应设备拥有的参数分区。由于所有的模型数据都被分区，在前向和后向传递中需要集体通信。在前向传递中，需要全部收集，以收集分区的模型权重，计算中间激活和模型输出。在后向过程中，模型权重被再次收集以计算梯度。分割后的梯度被还原到各自的设备中。由于每个设备只持有梯度和参数的分区，它将只更新分区的参数，而不是完整的模型参数。由于这种方法是对数据并行训练的加强，它与模型并行保持正交，可以与模型并行整合，以支持更大的模型。

D.异质性训练

在努力更好地利用GPU的计算能力的同时，人们往往忽略了CPU内存的容量一般比GPU内存大。因此，CPU内存对于打破深度学习训练中的内存限制至关重要。例如，在一个有8个Tesla V100 GPU的典型Nvidia DGX1工作站上，聚合的

每个节点的GPU内存为256GB，而CPU内存为512GB。如果CPU内存可以用于模型训练，那么模型大小可以简单地扩大。此外，由于NVMe固态硬盘（SSD）与传统硬盘（HDD）相比具有相对较高的通信带宽，NVMe固态硬盘也可以用来容纳模型数据，从而进一步增加单个节点的训练容量。

DeepSpeed提出了零卸载[[34]](#page13)，在不使用时将十进制从GPU移到CPU或NVMe磁盘上，以便为更大的模型腾出空间。通过利用高性能的异构存储设备，并在不同的硬件设备之间不断交换十进制，在单个GPU上训练一个具有数十亿参数的模型成为可能。这对计算资源有限的用户特别友好，对大规模模型训练的民主化至关重要。

1. 设计

随着过去十年计算资源的快速发展，各超级计算中心和研究实验室的硬件规格也变得多样化。这使得我们很难有一种通用的方法，由于硬件条件的巨大差异，对深度学习训练产生持续满意的加速。Megatron-LM和DeepSpeed也可以满足这样的问题。相比之下，Colossal-AI的特点是以模块化的方式构建了一系列的加速技术。这个加速技术库可以涵盖广泛的训练设置，以达到最大的性能。在本节中，我们将讨论Colossal-AI中集成的加速技术的实现和分析。

|  |  |  |
| --- | --- | --- |
| 模式 | 总通信量 | |
|  | / | 元素数 |
|  | 转移 | |
|  |  |  |
|  |  |  |
| 1D | 2(p | 1) Sx |
| 2D | 3(j 1) (Sx + S )w | |
| 2.5D | 3(k 1) (Sx =d + S )w | |
| 3D | 2(l 1)=l (Sx + Sw + S )y | |

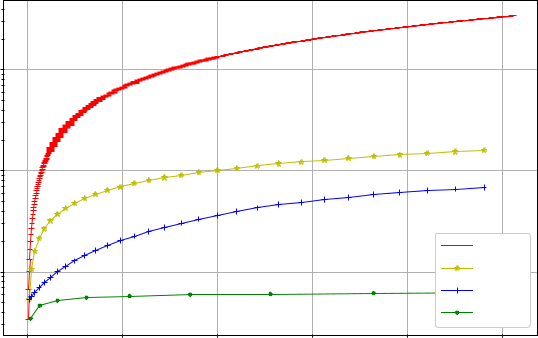
表一：实验的系统规范

A.多维模型并行化

如背景部分所述，模型并行是分布式训练中的一项重要技术，因为它减少了单个GPU的内存负担。因此，随着设备数量的增加，它可以使模型规模扩大到数十亿个参数。Colossal-AI提供了一系列的模型并行方法来满足分布式训练的需要。

Transformer模型的一个主要特点是其架构严重依赖线性层。由于张量并行主要应用于矩阵-矩阵乘法，它非常适用于Transformer模型训练的加速。在Colossal-AI中，所有现有的张量并行化方法都是

**按GPU数量计算的通信量**



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **量** | **10** |  |  |  |  |  |
| **10** |  |  |  |  |  |
|  |  |  |  |  |  |
| **沟通** | **9** |  |  |  |  |  |
| **10** |  |  |  |  |  |
|  |  |  |  |  | **1D** |
|  |  |  |  |  |  |
|  | **8** |  |  |  |  | **2D** |
|  | **10** |  |  |  |  |  |
|  |  |  |  |  |  | **2.5D** |
|  |  |  |  |  |  | **3D** |
|  | **0** | **200** | **400** | **600** | **800** | **1000** |
|  |  |  | **GPU的数量** | |  |  |

图9：理论分析中张量并行的扩展性能（h=1024；s=512；b=32）。

支持，这样用户就可以根据他们的训练要求和GPU的数量来选择一种方法。

在所有张量并行方法中，高级张量并行，即二维、2.5维和三维张量并行的一个突出优点是，与一维张量并行相比，它的通信成本较低。[表一](#page6)显示了计算矩阵乘法Y=W X时的总通信量，其中X的形状是（b；s；h），W的形状是（h；h），Y的形状是（b；h）。在表[一中，](#page6)使用了以下符号。

|  |  |
| --- | --- |
|  | p：GPU的总数量 |
|  | j：正方形的一边的GPU数量。 |
|  | 网络拓扑结构如图[6b](#page5)所示[，](#page5)其中p = j2 |
|  | k:在前广场一侧的GPU的数量。 |
|  | 正方形的网络拓扑结构，如图[6c](#page5)所示[、](#page5) |
|  | 其中p = d k2 |
|  | d：深度维度的GPU数量。 |
|  | 如图所示，长方体形的网络拓扑结构。 |
|  | 图[6c](#page5) |
| l：立方体的一侧的GPU数量。 | |
|  | 网络拓扑结构如图[6d](#page5)所示[，](#page5)其中p=l3 |
|  | Sx ：输入矩阵X中的元素数，。 |

除了张量并行，Colossal-AI还包括序列并行和管道并行，这样就有了混合并行来加速大规模集群中的模型训练。

B.增强的分片和卸载

DeepSpeed提供了对零冗余训练和卸载的支持。零冗余训练通过对模型参数、梯度和优化器状态进行分区来消除内存冗余。卸载利用CPU内存和NVMe磁盘来扩大大规模模型的可用存储空间。然而，不必要的复杂实现使得DeepSpeed的效率和可扩展性降低。因此，我们重新设计了分片和卸载模块，以提高性能。对于零冗余数据的并行训练，我们定义了一个统一的分片张量接口。张量的分区可以由可定制的分片策略来支持。这允许模型数据和优化器数据以松散耦合的共同方式分片。训练工作流程也可以通过用户定义的生命周期钩子轻松修改。通过这种方式，我们不仅可以实现DeepSpeed的零冗余机制，还可以为不同粒度的更多优化腾出空间。在Colossal-AI中，由于可扩展的设计，我们整合了PatrickStar[[33]](#page13)为内存空间提出的额外优化。

第一个优化是，我们重新使用内存中的fp16存储空间，以便可以训练更大的模型。在前向传递中，我们持有fp16参数，没有梯度。在后向传递中，fp16参数被用来计算梯度。当梯度被完全计算出来后，就不再需要fp16参数了。那么我们可以在前向和后向传递过程中把fp16梯度保存在保存fp16参数的存储空间中。这个过程是以子模块的粒度进行流式计算的。这可以进一步减少冗余和峰值内存的使用，如图[10](#page7)所示[。](#page7)通过这种方式，CPU的内存可以承受更大的模型。



同样的语义适用于SW 和 SY 。

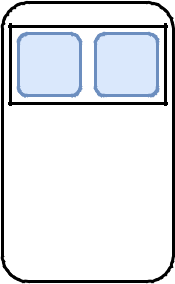
b：输入矩阵X的批量大小。

s：输入矩阵X的序列长度。

h：权重W的隐藏大小。

如图[9](#page7)所示[，](#page7)高级张量并行的通信量明显低于一维张量并行，特别是在使用大量节点时。通信效率的根本原因是高级张量并行只在计算节点的一个子组上产生通信。例如，在二维并行中，集体通信只涉及方形网络的一行或一列的节点。在

**前进**

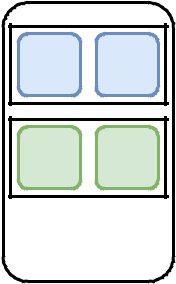


**P1** **P2**

**记忆**

**空间**

**向后**



**P1** **P2**

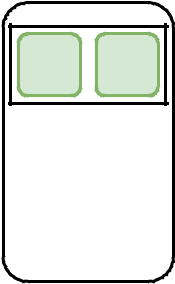
**G1** **G2**

**记忆**

**空间**

**岗位**

**向后**



**G1** **G2**

**记忆**

**空间**

相比之下，一维张量并行涉及所有计算节点的一个集体通信调用。因此，高级张量并行允许超过一个节点的扩展，而一维张量并行往往被限制在节点内计算。

图 10: 内存空间再利用

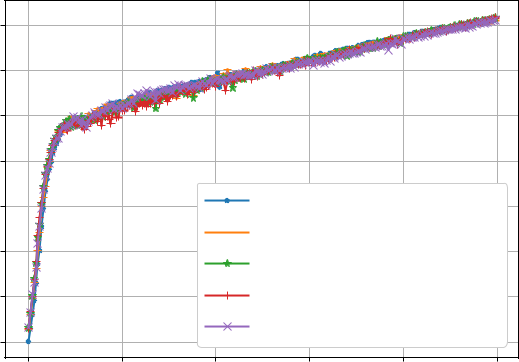
第二个优化是自适应混合亚当优化器。DeepSpeed在优化器的操作中使用CPU Adam，当

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 启用floading。所有的参数都被卸载和更新 3 | | | | | |  |  |
| 平行 = dict( |
| 在CPU内存中。相比之下，Colossal-AI的混合亚当4 | | | | | | tensor=dict(size=4, mode='1d') |  |
| 优化器监控GPU上的可用内存空间。 5 | | | | | | ) |  |
| 清单1：配置文件 | |
| 它将在GPU上保留部分参数和梯度作为 | | | | | |
| 只要有剩余的空间。通过这种方式，参数被更新 | | | | | | 3）用户友好性：为了尽量减少对用户的改变 | |
| 在CPU和GPU上，导致了较低的时间成本。 | | | | | |
|  | 由于卸载需要在不同的张量之间交换 | | | | | 代码，Colossal-AI为模型训练提供了用户友好的API。 | |
| 通过PCIe的设备，会有额外的通信开销。 | | | | | | ing.用户只需要准备一个配置文件作为 | |
| 引起的。然而，这创造了更多的可用内存空间 | | | | | | 在清单[1](#page8)中显示。该配置文件是一个简单的Python | |
| 用于模型训练。这本质上带来了两个好处，1) | | | | | | 文件，该文件通过遵循预先定义的 | |
| 在输入数据大小不变的情况下，更大的模型可以是 | | | | | | 模式。然后，Colossal-AI将注入加速功能 | |
| 在GPU中被容纳，更大的模型可以带来更好的 | | | | | | 用'colossalai.initialize'进入执行引擎，如图所示 | |
| 预测性能。2）给定一个恒定规模的模型、 | | | | | | 在列表[1](#page8)中[。](#page8) | |
| 可以使用较大的批处理量，以便训练产量能够 | | | | | | Colossal-AI还提供了并行化的流行模型com-。 | |
| 尽管在PCIe上有额外的通信，但还是会有改进。 | | | | | | 用户可以直接在方便的情况下使用这些组件。这也是 | |
|  |  | IV.执行情况 | | | | 不要求用户拥有领域的专业知识，以便他们能够 | |
|  |  | 不需要再像以前那样手动设计其并行策略。 | |
|  | Colossal-AI是一个深度学习系统，用以下方式实现 | | | | |
|  | GShard[13]。 | |
| PyTorch。Colossal-AI的整体架构如下图所示 | | | | | |  |  |
| 图1.Colossal-AI的基本组成部分是 1 | | | | | | 进口巨无霸 |  |
|  |  |  |  | 2 | |  |  |
| 并行上下文管理器。它负责存储元3 | | | | | |  |  |
| # 启动分布式网络 |  |
| 关于复杂的混合并行分布式系统的信息... 4 | | | | | | colossalai.launch\_from\_torch(config='./config.py') |  |
| 诸如分布式网络拓扑结构等环境。基于 5 | | | | | |  |  |
| # 确定你的培训内容 |  |
|  |  |  |  | 6 | |  |
| 并行情况下，分布式运算符会自动7 | | | | | | ... |  |
| 切换到相应的并行模式。例如，如果8 | | | | | |  |  |
| 并行环境指定使用一维张量并行，则 9 | | | | | | #用Colossal-AI初始化 |  |
|  |  |  |  | 10 | | engine, trainloader, testloader, lr\_scheduler = = = = = = |  |
| 线性层的计算流程将被修改，以包括11个 | | | | | |  |  |
| colossalai.initialize(model、 |  |
| 分行和分列的矩阵乘法，如12 | | | | | | 优化器、 |  |
| 图[4](#page4)在运行时。在此 | | | | 方式，它很容易建立 13 | | 标准、 |  |
| 装火车的人、 |  |
| 一个张量并行的模型，很容易成为 | | | | 14 | |  |
| 复杂的计算 15 | | testloader、 |  |
| 和通信工作流程对用户是隐藏的，16 | | | | | | lr\_scheduler) |  |
| 从模型建立过程中分离出来。用户可以 17 | | | | | | #跑步训练 |  |
|  |  |  |  | 18 | |  |
| 使用这些运算符，就像使用PyTorch的本地运算符来构建19 | | | | | |  |  |
| for data, label in train\_dataloader： |  |
| 他们的模型将执行并行计算图 20 | | | | | | data = data.cuda() |  |
| 在运行期间。上面一层 | | | | 在各种加速器中 21 | | label = label.cuda() |  |
| 引擎.zero\_grad() |  |
|  |  |  |  | 22 | |  |
| 诊断工具。基本工具包括激活检查点、23 | | | | | | output = engine(data) |  |
| 混合精确训练等等。更高级的将24 | | | | | | train\_loss = engine.criterion(output, label) |  |
| 对大规模模型进行优化分片和卸载。 25 | | | | | | 引擎.后退(train\_loss) |  |
|  |  |  |  | 26 | | 引擎.步骤() |  |
| 训练。最后，执行引擎负责控制27 | | | | | |  |  |
| lr\_scheduler.step() |  |
| 培训过程中，培训师提供了一个更高层次的APIs | | | | | | 清单2：Colossal-AI的用法 | |
| 来开始训练的线路。这两个组成部分提供了巨大的 | | | | | |
|  |  |
| 用户自定义的可扩展性，因为他们可以定义他们的 | | | | | |  |  |
| 自己的 | | 培训计划 | 和钩子 | 在操作员或培训师 | | V.评估 | |
| 水平。 | |  |  |  |  |
|  |  |  |  |  |  |
| 1) | | 模块化： ǞǞǞ | 模块化和扩展性的原则 | | | A.实验设置 | |
| 在整个发展过程中，我们都坚持了可持续发展的原则。 | | | | | |
| 为了全面评估Colossal-的系统性能。 | |
| 加速技术可以很容易地结合起来，以追求 | | | | | |
| 最大的性能。 | | |  |  |  | AI，我们已经在不同的硬盘上进行了各种实验。 | |
| 2) | | 可扩展性：作为一个不断发展的系统、 | | | | 器。系统规格列于[表二。](#page9)由于 | |
| Colossal-AI提供了各种接口来实现客户的需求。 | | | | | | 由于资源的限制，我们只测试了一部分著名的 | |
| 为今后的扩展提供了专门的功能。例如， | | | | | | 实验项目中所说的每个系统的特征 | |
| 分片模块允许用户定义自己的分片。 | | | | | | 列。 | |
| 战略和生命周期挂钩，以试图探索 | | | | | | 我们使用Megatron-LM和DeepSpeed作为我们的基线 | |
| 更有效的培训方法。 | | | |  |  | 用于实验。由于Megatron-LM的张量并行性已被 | |
|  |  |  |  |  |  | 在Colossal-AI中实现，它将被注释为一维张量 | |
| 1 | # 指定使用一维张量并行制 | | | |  |
|  | 实验结果中的平行性。 | |
| 2 | # 有着并行的大小4 | | |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 系统 | #GPU | #节点 | GPU模型 | GPU互连 | 跨节点互连 | | | 实验项目 | |
| 身份证 | 每 |  |  |  |  |  |  |  |  |
|  | 结点 |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | |
| I | 8 | 1 | Nvidia A100 (80GB) | 联网 | 不适用 |  |  | 张量并行制 | |
| 二 | 8 | 1 | Nvidia A100 (80GB) | 在adja-之间的NV链接 | 不适用 |  |  | 张量并行制, ZeRO | |
|  |  |  |  | 芯片，PCIe，以及其他一些产品。 |  |  |  |  |  |
|  |  |  |  | 遥远的GPU之间的联系 |  |  |  |  |  |
| 三 | 4 | 16 | Nvidia A100 (40GB) | 视觉效果 | 英飞尼迪宽带 | | HDR | 张量 | 平行主义、 |
|  |  |  |  |  | (200Gb/s)，以及Dragonfly | | | 序列平行化 | |
|  |  |  |  |  | 网络拓扑结构 | |  |  |  |
| 四 | 1 | 64 | Nvidia P100 (16GB) | RDMA | Cray | 白羊座的路由和 | | 张量并行制 | |
|  |  |  |  |  | 通讯 | | ASIC、 |  |  |
|  |  |  |  |  | 和 | 蜻蜓 | 网络 |  |  |
|  |  |  |  |  | 拓扑学 | |  |  |  |

表二：实验的系统规范

**ViT在ImageNet-1K上的收敛性**



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **0.7** |  |  |  |  |  |
|  | **0.6** |  |  |  |  |  |
| **准确度** | **0.5** |  |  |  |  |  |
| **0.4** |  |  |  |  |  |
| **0.3** |  |  | **非张量并行** | | |
|  | **0.2** |  |  | **1D** |  |  |
|  |  |  | **2D** |  |  |
|  |  |  |  |  |  |
|  | **0.1** |  |  | **2.5D** |  |  |
|  | **0.0** |  |  | **3D** |  |  |
|  |  |  |  |  |  |
|  | **0** | **50** | **100** | **150** | **200** | **250** |
|  |  |  | **纪元** | |  |  |

图11：ViT在ImageNet上的收敛性能

B.多维张量并行化

1) 收敛

我们在ImageNet-1k数据集上用Vision Transformer（ViT）[[2]](#page12)进行了实验，以验证系统III上多维张量并行的算术正确性和数值稳定性。ViT模型被配置为有12个Transformer层，隐藏大小为384，6个注意头。我们使用Jax初始化和AdamW优化器，学习率为0.003，重量衰减为0.3。输入图像的形状为224，补丁大小为16。全局批次大小为4k，模型训练了250个epochs。基准是用数据并行的方式进行模型训练，因为每个设备都保持完整的计算图。如图[11](#page9)所示[，](#page9)多维张量并行的测试精度曲线与基线一致，表明多维张量并行不影响模型的收敛。

2) 内存效率

由于二维、2.5维、三维张量并行划分了输入数据、层权重和输出激活，而一维张量并行只划分了层权重，预计前者的内存消耗较低。因此，前三个

这些方法使GPU能够容纳更大的模型。为了证明内存效率，我们在系统I上进行了两个范围测试，通过批处理大小和隐藏大小进行扩展。在这个范围测试中，我们创建了一个由两个线性层组成的模型。我们在4个GPU上运行1D、2D和2.5D实验，在8个GPU上运行1D、2.5D（深度=2）和3D实验。我们测量了前向和后向传递过程中分配的最大CUDA内存，结果显示在图[12](#page10)中[。](#page10)一维张量并行的内存消耗远远高于二维、2.5维和三维张量并行。在批量大小等于512和8个GPU的情况下，2.5D和3D的内存消耗分别比图[12b](#page10)中的1D张量并行低44%和65%[。](#page10)在隐藏大小为16384和8个GPU的情况下，2.5D和3D张量并行的内存性能分别比图[12d](#page10)中的1D张量并行好62%和74.2%[。](#page10)因此，在扩展到超大规模模型时，更高级的张量并行是一个更好的选择。

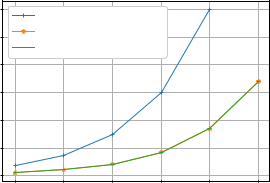
3) 硬件兼容性

为了进一步探索各种张量并行方法的性能，我们在系统I和系统II上进行了实验，研究GPU互连对张量并行性能的影响。系统I和系统

1. 被选作实验，因为如图[5a](#page4)所示，前者在任何一对GPU之间都有完全连接的NVLink，而后者在4对相邻的GPU之间只有NVLink，如图[5b](#page4)所示[。](#page4)如图[13](#page10)所示，无论对一对GPU还是一组GPU进行测量，系统I的通信带宽都一直很高。然而，当通信发生在非相邻的GPU之间时，通信带宽从184GB/s大幅下降到15GB/s，因为只有相邻的GPU有高性能的NVLink。

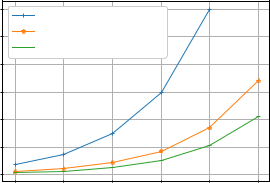
因此，系统II的GPU拓扑结构对1D张量并行化并不友好，因为1D张量并行化依赖于通过PCIe在所有GPU之间进行全还原操作。相反，2D和2.5D只有一对GPU之间的通信，而不是跨越所有GPU。这允许部分通信仍然利用相邻GPU之间的高NVLink带宽。

**最大分配的内存**



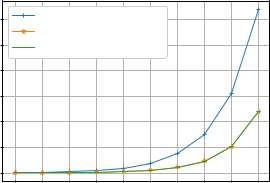
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **60** | **1d (ws=4, tp=4)** | |  |  |  |
|  |  |  |  |  |
|  | **50** | **2d (ws=4, tp=4)** | |  |  |  |
|  | **2.5d (ws=4, tp=4)** | | |  |  |
| **内存/GB** |  |  | **内存/GB** |
| **40** |  |  |  |  |
| **30** |  |  |  |  |
| **20** |  |  |  |  |
|  |  |  |  |  |  |
|  | **10** |  |  |  |  |  |
|  | **0** |  |  |  |  |  |
|  | **5** | **6** | **7** | **8** | **9** | **10** |
|  | **2** | **2** | **2** | **2** | **2** | **2** |
|  |  |  | **批量大小** | |  |  |

**最大分配的内存**



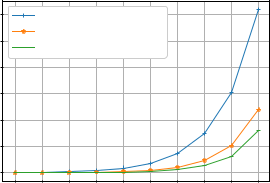
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **60** | **1d (ws=8, tp=8)** | |  |  |  |
|  |  |  |  |
| **50** | **2.5d (ws=8, tp=8)** | | |  |  |
| **3d (ws=8, tp=8)** | |  |  |  |
|  |  |  |  |
| **40** |  |  |  |  |  |
| **30** |  |  |  |  |  |
| **20** |  |  |  |  |  |
| **10** |  |  |  |  |  |
| **0** |  |  |  |  |  |
| **5** | **6** | **7** | **8** | **9** | **10** |
| **2** | **2** | **2** | **2** | **2** | **2** |
|  |  | **批量大小** | |  |  |

**最大分配的内存**



|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **60** |  | **1d (ws=4, tp=4)** | | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  | **2d (ws=4, tp=4)** | | | |  |  |  |  |
| **内存/GB** | **50** |  | **2.5d (ws=4, tp=4)** | | | |  |  |  |  |
| **40** |  |  |  |  |  |  |  |  |  |
| **30** |  |  |  |  |  |  |  |  |  |
| **20** |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  | **10** |  |  |  |  |  |  |  |  |  |
|  | **0** |  |  |  |  |  |  |  |  |  |
|  | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
|  | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** |
|  |  |  |  |  | **隐藏的尺寸** | | |  |  |  |

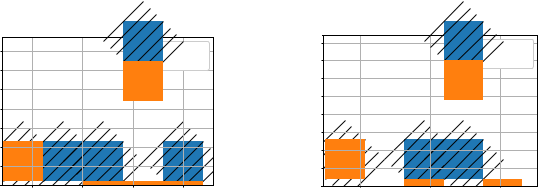
**最大分配的内存**



|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **60** | **1d (ws=8, tp=8)** | | | | |  |  |  |  |
|  | **50** | **2.5d (ws=8, tp=8)** | | | | |  |  |  |  |
| **内存/GB** | **3d (ws=8, tp=8)** | | | | |  |  |  |  |
|  |  |  |  |  |
| **40** |  |  |  |  |  |  |  |  |  |
| **30** |  |  |  |  |  |  |  |  |  |
| **20** |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  | **10** |  |  |  |  |  |  |  |  |  |
|  | **0** |  |  |  |  |  |  |  |  |  |
|  | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
|  | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** |
|  |  |  |  |  | **隐藏的尺寸** | | |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (a) 按批量大小划分的4个GPU | | | (b) 按批量大小划分的8个GPU | | (c) 4个GPU的隐藏尺寸 | | (d) 按隐藏尺寸划分的8个GPU | | |
|  |  | 图12：4/8个GPU的张量并行化的内存消耗范围测试 | | | | | | |  |
|  |  |  |  |  |  |  |  |  |  |
|  | #GPU | 模式 | #转换机层 | 隐藏的尺寸 | #Attention Head | 批量大小 | 吞吐量 |  | 比1D的速度更快 |
|  |  |  |  |  |  |  | (img/sec) |  | (%) |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | 1D |  |  |  | 128 | 5.06 |  | - |
|  | 4 | 2D | 24 | 2048 | 32 | 256 | 6.18 |  | 22.1 |
|  |  | 2.5D (深度=1) |  |  |  | 256 | 6.73 |  | 33.0 |
|  |  | 1D |  |  |  | 256 | 7.46 |  | - |
|  | 8 | 2.5D (深度=2) | 24 | 2048 | 32 | 384 | 6.57 |  | -11.9 |
|  |  | 3D |  |  |  | 512 | 8.38 |  | 12.3 |
|  |  | 1D |  |  |  | 64 | 3.42 |  | - |
|  | 16 | 2D | 32 | 4096 | 64 | 256 | 5.33 |  | 55.8 |
|  |  | 2.5D (深度=1) |  |  |  | 256 | 5.46 |  | 59.6 |
|  |  | 1D |  |  |  | 128 | 4.22 |  | - |
|  | 32 | 2.5D (深度=2) | 32 | 4096 | 64 | 256 | 5.46 |  | 50.6 |
|  |  | 1D |  |  |  | 128 | 4.63 |  | - |
|  | 64 | 2D | 32 | 4096 | 64 | 512 | 12.76 |  | 275.5 |
|  | 2.5D (深度=4) | 512 | 4.93 |  | 6.5 |
|  |  |  |  |  |  |
|  |  | 3D |  |  |  | 512 | 8.63 |  | 86.4 |
|  |  |  |  |  |  |  |  |  |  |

表三：不同数量的GPU的张量并行的性能



**两个GPU之间的带宽**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **175** |  |  |  | **系统一** |
|  |  |  |  |  |
| **(GB/s)** |  |  |  |  | **系统二** |
| **150** |  |  |  |  |
| **125** |  |  |  |  |
|  |  |  |  |  |
| **带宽** | **100** |  |  |  |  |
| **75** |  |  |  |  |
| **50** |  |  |  |  |
|  |  |  |  |  |
|  | **25** |  |  |  |  |
|  | **0** | **0-1** | **0-2** | **0-4** | **0-6** |
|  |  |
|  |  |  | **GPU对** | |  |

1. GPU对之间的通信带宽

**集体通信带宽**

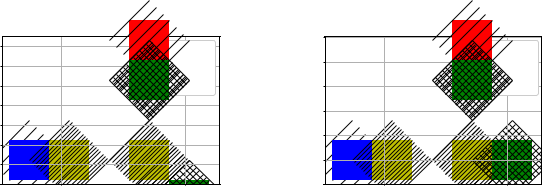
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **200** |  |  | **系统一** |
|  |  |  |  |
| **(GB/s)** | **175** |  |  | **系统二** |
| **150** |  |  |  |
|  |  |  |  |
| **带宽** | **125** |  |  |  |
| **100** |  |  |  |
| **75** |  |  |  |
| **50** |  |  |  |
|  |  |  |  |
|  | **25** |  |  |  |
|  | **0** | **2** | **4** | **8** |
|  |  |
|  |  |  | **GPU的数量** |  |

1. 集体通信的通信带宽

由于有更多的内存可用，隐藏的大小和注意头的数量分别被调整为4096和64。

该模型以不同的批次大小设置进行训练，直到出现内存不足的问题。因此，我们提出了每种张量并行方法的最佳吞吐量。在图[14a](#page10)中[，](#page10)二维、2.5维和三维张量并行的吞吐量在4个GPU和8个GPU上都无法与一维张量并行竞争。这是预期的，有两个原因。第一个原因是，一维张量并行可以利用

图13：系统I和II的通信带宽（使用NCCL带宽测试工具广播125MB数据）。



第二个原因是，二维、二点五维和三维张量并行在少数处理器上有更多的通信量，并且只会超过一维张量。

**ViT的吞吐量 - 系统I**

|  |  |  |  |
| --- | --- | --- | --- |
| **(img/sec)** | **35** |  | **1D** |
| **30** |  | **2D** |
|  | **2.5D** |
|  |  |
| **25** |  | **3D** |
|  |  |  |
| **吞吐量** | **20** |  |  |
| **15** |  |  |
| **10** |  |  |
|  |  |  |
|  | **5** |  |  |
|  | **0** | **4** | **8** |
|  |  |

**GPU的数量**

(a) 系统I

|  |  |  |  |
| --- | --- | --- | --- |
|  | **30** | **ViT-系统II的吞吐量** | |
|  |  |  |
| **(img/sec)** |  |  | **1D** |
| **25** |  | **2D** |
|  |  | **2.5D** |
| **20** |  | **3D** |
|  |  |  |
| **吞吐量** | **15** |  |  |
| **10** |  |  |
| **5** |  |  |
|  |  |  |
|  | **0** | **4** | **8** |
|  |  |

**GPU的数量**

(b) 系统二

当处理器的数量增加时，并行性也随之增加。然而，当实验被切换到系统II时，在

图[14b，](#page10)一维张量并行将遇到瓶颈，因为在4个和8个GPU之间的集体通信带宽很低。同时，2D和2.5D可以提供比4个GPU的1D张量并行高40%的吞吐量。在有8个GPU的情况下，2.5D张量并行仍能比1D张量并行的吞吐量高40%。

图14：系统I和II上ViT训练的吞吐量

20:6%.由于扩展性低，三维张量并行的性能仍然低于一维张量并行。

我们在ImageNet-1k数据集上对ViT进行了训练，在系统I和系统8的不同配置下，4个GPU和8个GPU都进行了训练。

1. 在4个GPU上，ViT模型有64个Transformer层，隐藏大小为3072，有48个注意头。在8个GPU上，

4) 吞吐量比较

为了测试更多GPU的张量并行性能，我们在系统IV上训练ViT。由于系统IV只有16GB的GPU内存，因此，我们调整了

ViT模型的配置也是如此。对于4和8个GPU，该模型被设置为24层，隐藏大小为2048，32个注意头。从16个GPU开始，模型被设置为32层，隐藏大小为4096，注意头为64。

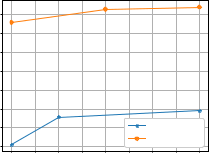
4到64个GPU的结果显示在表[三中。](#page10)可以看出，随着GPU数量的增加，高级张量并行方法比一维张量并行方法的速度提高了2.76。这可以归因于高级张量并行方法在扩展到更多处理器时的较低通信量。加上内存效率和低通信量，二维、2.5维和三维张量并行是大规模分布式训练的更好选择。

12个GPU的一维张量并行。在最大序列长度测试中也可以观察到同样的情况，如图[15b](#page11)所示[，](#page11)这再次证实了序列并行主义在内存效率方面的优势。序列并行主义的最大序列长度是序列并行主义的1.18倍。由于BERT-Base模型中的自我注意模块相对于序列长度来说是二次复杂的，最大序列长度只能随着GPU数量的增加而呈次线性扩展。如果使用线性复杂度的自我关注模块[[36], [37]](#page13)，序列并行主义可以实现最大序列长度与GPU数量的线性扩展，更好地支持文档级文本理解。

C.序列并行化

在本节中，我们将序列并行与一维张量并行进行比较，以证明其在内存效率和训练吞吐量方面的优势。由于序列并行主义是为激活所消耗的内存大于模型数据所消耗的内存的情况而设计的，因此选择BERT-Base作为我们的实验模型，并训练了

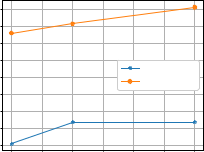
**BERT-Base的训练吞吐量**



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **(tokens/sec)** | **90000** |  |  |  |  |  |  |  |  |
| **85000** |  |  |  |  |  |  |  |  |
| **80000** |  |  |  |  |  |  |  |  |
| **75000** |  |  |  |  |  |  |  |  |
| **吞吐量** |  |  |  |  |  |  |  |  |
| **70000** |  |  |  |  |  |  |  |  |
| **65000** |  |  |  |  |  |  |  |  |
| **60000** |  |  |  |  |  | **1D** |  |  |
|  | **55000** |  |  |  |  |  | **序列** | | |
|  |  |  |  |  |  |  |  |  |
|  | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
|  |  |  |  | **GPU的数量** | | | |  |  |

(a) 没有管道的情况下

**具有管道并行性的BERT-BASE的训练吞吐量**



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **(tokens/sec)** | **95000** |  |  |  |  |  |  |
| **90000** |  |  |  |  |  |  |
| **85000** |  |  |  |  |  |  |
| **80000** |  |  |  |  | **1D** |  |
| **吞吐量** |  |  |  |  |  |  |
| **75000** |  |  |  |  | **序列** | |
|  |  |  |  |  |
| **70000** |  |  |  |  |  |  |
| **65000** |  |  |  |  |  |  |
| **60000** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | **55000** |  |  |  |  |  |  |
|  | **1.0** | **1.5** | **2.0** | **2.5** | **3.0** | **3.5** | **4.0** |
|  |  |  | **管道尺寸（PIPELINE\_SIZE** | | |  |  |

(b) 有管道

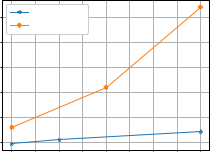
在维基百科的数据集上[[35]。](#page13)我们在系统III上进行了实验。需要注意的是，一维张量并行需要注意头的数量（12个）被并行规模整除，我们只能使用4、6和12个GPU，其中6-GPU的实验使用2个节点，每个节点使用3个GPU。同时，序列并行主义不受注意力头数的限制，因此我们在4、8和12个GPU上进行实验。

图 16: BERT-Base的训练吞吐量

2) 吞吐量比较

为了评估序列并行的训练速度，我们以512的序列长度和其最大的批次大小训练了BERT-Base。吞吐量的指标是每秒钟的令牌。如图[16a](#page11)所示[，](#page11)序列并行主义一直提供较高的训练吞吐量，序列并行主义的吞吐量是1D的1.43倍。

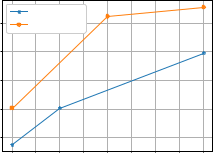
**BERT-Base的最大批量大小**



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1200** | **1D** |  |  |  |  |  |  |  |
|  | **序列** | | |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **尺寸** | **1000** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| **批量** | **800** |  |  |  |  |  |  |  |  |
| **600** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  | **400** |  |  |  |  |  |  |  |  |
|  | **200** |  |  |  |  |  |  |  |  |
|  | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
|  |  |  |  | **GPU的数量** | | | |  |  |

(a) 最大批处理量

**BERT-Base的最大序列长度**



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **1D** |  |  |  |  |  |  |  |
| **长度** | **2000** | **序列** | | |  |  |  |  |  |
| **1800** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| **序列** | **1600** |  |  |  |  |  |  |  |  |
| **1400** |  |  |  |  |  |  |  |  |
|  | **1200** |  |  |  |  |  |  |  |  |
|  | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
|  |  |  |  | **GPU的数量** | | |  |  |  |

(b) 最大序列长度

张量并行化。

正如在设计部分所提到的，序列并行主义与管道并行主义自然兼容，不需要在管道阶段之间进行额外的通信。我们进一步扩大了与管道并行主义的训练规模。序列并行和一维张量并行的并行规模都固定为4，我们将流水线阶段的数量从1扩展到4。 如图[16b](#page11)所示[，](#page11)序列并行可以训练1.55

图 15: 序列并行比一维张量并行的内存效率

倍，比1D张量并行的4个流水线阶段快，这为非流水线结果提供了12%的额外加速。

1) 内存效率

为了说明序列并行在BERT训练中的内存效率，我们增加了批次大小和序列长度，直到1D张量并行和序列并行都出现内存不足的问题。在批量大小测试中，序列长度被固定为512，而在序列长度测试中，批量大小被固定为64。

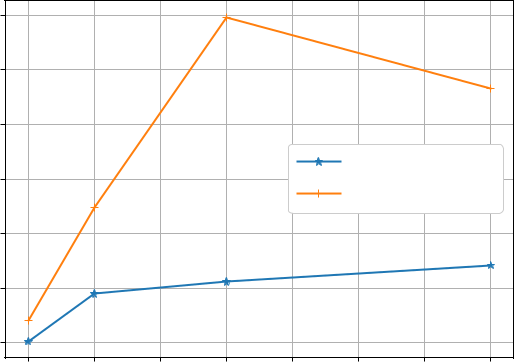
如图[15a](#page11)所示[，](#page11)序列平行主义可以持续达到比一维张量平行主义更大的批次规模。这是因为一维张量并行在重复激活方面有内存瓶颈，而在序列并行中，激活是沿序列维度分割的。序列并行主义的最大批处理量比1D张量并行主义大4.44倍。

D.分片和卸载

在这一部分，我们针对DeepSpeed评估了我们自己的分片和卸载模块。我们使用DeepSpeed第3阶段作为基线，它在数据并行训练中分割模型参数、梯度和优化器状态。Colossal-AI和DeepSpeed都将优化器数据卸载到CPU内存中。我们在系统II上对维基百科数据集进行了100亿个参数的GPT-2模型训练。我们将数据并行训练的规模从1个GPU扩大到8个GPU，并观察批量大小等于4时的训练吞吐量。

如图[17](#page12)所示，在[III-B](#page7)节讨论的额外优化下[，](#page7)Colossal-AI在8个GPU上比DeepSpeed快2.33倍。

**GPT的吞吐量**



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **(tokens/sec)** | **700** |  |  |  |  |  |  |  |
| **600** |  |  |  |  |  |  |  |
| **500** |  |  |  |  |  |  |  |
|  |  |  |  |  | **深速** | | |
| **吞吐量** | **400** |  |  |  |  |
|  |  |  |  | **巨大的AI** | |  |
|  |  |  |  |  |  |
| **300** |  |  |  |  |  |  |  |
| **200** |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | **100** |  |  |  |  |  |  |  |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|  |  |  | **GPU的数量** | | | |  |  |

图 17: 带有分片和卸载的GPT训练的吞吐量

VI.结论

在这项工作中，我们设计并实现了Colossal-AI，它将大量的先进加速技术整合到一个统一的系统中，用于大规模的分布式训练。Colossal-AI具有灵活的系统设计，支持不同并行方法的组合，并允许开发人员和用户轻松扩展。此外，它的加速技术在不同的硬件条件下提供了强大的性能，并且与基线系统相比提供了卓越的性能。在我们的实验中，我们已经证明了Colossal-AI比Megatron-LM和DeepSpeed的优势。与Megatron-LM的1D张量并行相比，当2层MLP的隐藏大小扩展到16384时，2.5D和3D张量并行可以节省高达62%和74.2%的内存使用。当扩展到64个GPU时，二维张量并行可以超过Megatron-LM一维张量并行的276%。对于具有大量激活的中等规模的模型，如BERT-Base，序列并行可以提供比1D张量并行高1.43倍的训练吞吐量。我们重新设计的分片和卸载模块可以比GPT-2上的DeepSpeed快2.33倍。

参考文献

1. J.Devlin, M.-W.Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," arXiv preprint arXiv: 1810.04805, 2018.
2. A.Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai、
   1. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly et al., "An image is worth 16x16 words：规模化图像识别的变换器"，arXiv预印本arXiv:2010.11929，2020。
3. A.Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners, " 2019.
4. T.B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal、
   1. Neelakantan, P. Shyam, G. Sastry, A. Askell等, "Language models are few-shot learners," arXiv preprint arXiv:2005.14165, 2020.
5. Z.Du, Y. Qian, X. Liu, M. Ding, J. Qiu, Z. Yang, and J. Tang, "All nlp tasks are generation tasks：A general pretraining framework," arXiv preprint arXiv:2103.10360, 2021.
6. J.Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization".机器学习研究杂志》，第12卷，第7期，2011年。
7. D.P. Kingma和J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.
8. K.He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp.770-778。
9. T.Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," 2016.[在线]。Available: <https://arxiv.org/abs/1604.06174>
10. Y.Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and z. Chen、

"Gpipe：Efficient training of giant neural networks using pipeline parallelism," in Advances in Neural Information Processing Systems, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alche´-Buc, E. Fox, and R. Garnett, Eds., vol. 32。Curran Associates, Inc., 2019。[在线]。Available: [https://proceedings.neurips.cc/paper/ 2019/file/093f65e080a295f8076b1c5722a46aa2-Paper.pdf](https://proceedings.neurips.cc/paper/2019/file/093f65e080a295f8076b1c5722a46aa2-Paper.pdf)

1. D.Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur、
   1. R.Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream：用于dnn训练的通用流水线并行性，"在第27届ACM操作系统原理研讨会论文集，Ser.SOSP '19.New York, NY, USA: Association for Computing Machinery, 2019, p. 1-15.[在线]。Available: <https://doi.org/10.1145/3341301.3359646>
2. M.Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catan-zaro, "Megatron-lm：使用模型并行性训练数十亿参数的语言模型，"arXiv预印本arXiv:1909.08053，2019。
3. D.Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun、
   1. Shazeer, and Z. Chen, "fGSghard: Scaling giant models with conditional computation and automatic sharding," in International Conference on Learning Representations, 2021.[在线]。可见[：https://openreview.net/forum?id=qrwe7XHTmYb](https://openreview.net/forum?id=qrwe7XHTmYb)
4. M.Baines, S. Bhosale, V. Caggiano, N. Goyal, S. Goyal, M. Ott、
   1. Lefaudeux, V. Liptchinsky, M. Rabbat, S. Sheiffer, A. Sridhar, and M. Xu, "Fairscale：用于高性能和大规模训练的通用模块化pytorch库，[" https://github.com/ facebookresearch/fairscale,](https://github.com/facebookresearch/fairscale) 2021.
5. D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary、
   1. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro、

A.Phanishayee, and M. Zaharia, "Efficient large-scale language model training on gpu clusters using megatron-lm, " in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ser.SC '21.美国纽约：计算机械协会，2021年。[在线]。Available: <https://doi.org/10.1145/3458817.3476209>

1. J.Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 1000 billion parameters," in Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp.3505- 3506.
2. D.Paperno, G. Kruszewski, A. Lazaridou, Q. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernandez, ´ "The lambada dataset：需要广泛话语背景的词汇预测，" 2016年6月，第1525-1534页。
3. D.Kingma和J. Ba, "Adam: A method for stochastic optimization," International Conference on Learning Representations, 12 2014.
4. A.Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N.

Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. 第30卷。Curran Associates, Inc., 2017。[在线]。Available: [https://proceedings.neurips.cc/paper/ 2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf](https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf)

1. Q.Xu, S. Li, C. Gong, and Y. You, "An efficient 2d method for training super-large deep learning models," arXiv preprint arXiv:2104.05343, 2021.
2. B.Wang, Q. Xu, Z. Bian, and Y. You, "2.5维分布式模型训练，" arXiv preprint arXiv:2105.14500, 2021.
3. Z.Bian, Q. Xu, B. Wang, and Y. You, "Maximizing parallelism in distributed training for huge neural networks," arXiv preprint arXiv:2105.14450, 2021.
4. L.E. Cannon, A cellular computer to implement the Kalman filter algorithm.蒙大拿州立大学，1969。
5. J.Berntsen, "超立方体上的通信效率矩阵乘法," Parallel computing, vol. 12, no.3, pp. 335-342, 1989.
6. R.A. van de Geijn和J. Watts，"Summa：可扩展的通用矩阵乘法算法"，美国，Tech.Rep., 1995.
7. E.Solomonik和J. Demmel, "通信优化的并行2.5d矩阵乘法和Lu分解算法", in Euro-Par, 2011.
8. R.C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar, "A three-dimensional approach to parallel matrix multiplication, " IBM Journal of Research and Development, vol. 39, no.5, pp. 575-582, 1995.
9. A.Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison、

A.Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy、

B. 斯泰纳 L. Fang、 J. Bai、 和 S. Chintala、 "Pytorch： 一个

势在必行 风格、 高性能 深度 学习 库，" 在

神经信息处理系统的进展32，H. Wallach、

H. Larochelle、 A. Beygelzimer、 F. d'Alche´-Buc、 E. Fox、 和

R. Garnett、 Eds. 库伦 Associates、 Inc.、 2019, pp. 8024-

8035.[在线].Available: [http://papers.neurips.cc/paper/9015-pytorch-](http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf)

[an-imperative-style-high-performance-deep-learning-library.pdf](http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf)

1. D.Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur、
   1. R.Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream：用于dnn训练的通用流水线并行性，"在第27届ACM操作系统原理研讨会论文集，Ser.SOSP '19.New York, NY, USA: Association for Computing Machinery, 2019, p. 1-15.[在线]。Available: <https://doi.org/10.1145/3341301.3359646>
2. Y.Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee、
   1. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, GPipe：使用管线并行性高效训练巨型神经网络。Red Hook, NY, USA: Curran Associates Inc., 2019.
3. S.Li和T. Hoefler, "Chimera：高效地训练具有双向管道的大规模神经网络，"在国际高性能计算、网络、存储和分析会议论文集，Ser.SC '21。美国纽约：计算机械协会，2021年。[在线]。Available: <https://doi.org/10.1145/3458817.3476145>
4. S.Li, F. Xue, Y. Li, and Y. You, "Sequence parallelism：从系统的角度看长序列训练，" 2021年。[在线]。Available: <https://arxiv.org/abs/2105.13120>
5. J.Fang, Y. Yu, Z. Zhu, S. Li, Y. You, and J. Zhou, "Patrickstar：通过基于分块的内存管理对预训练的模型进行并行训练，" 2021.[在线]。Available: <https://arxiv.org/abs/2108.05818>
6. J.Ren, S. Rajbhandari, R. Y.Aminabadi, O. Ruwase, S. Yang, M. Zhang、
   1. Li, and Y. He, "Zero-offload：民主化十亿规模的模型训练"，2021年。
7. W.基金会。维基媒体下载。[在线].Available: [https: //dumps.wikimedia.org.](https://dumps.wikimedia.org)
8. S.Wang, B. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer：具有线性复杂性的自我关注"，arXiv预印本arXiv:2006.04768，2020。
9. M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti、
   1. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed, "大鸟：较长序列的变形器，"在《神经信息处理系统进展》中，H. Larochelle, M. Ranzato, R. Hadsell、
   2. F.Balcan, and H. Lin, Eds., vol. 33.Curran Associates, Inc., 2020, pp.17 283-17 297.[在线]。Available: [https://proceedings.neurips.cc/ paper/2020/file/c8512d142a2d849725f31a9a7a361ab9-Paper.pdf](https://proceedings.neurips.cc/paper/2020/file/c8512d142a2d849725f31a9a7a361ab9-Paper.pdf)