TAPS：面向深度神经网络的拓扑感知算子内并行策略搜索算法

Peng Liang、Hao Zheng、 *Teng Su、Linbo Qiao、
Dongsheng Li 国防科技大学、 *华为技术有限公司 {peng_leung,
zhengh, qiao.linbo, dsli}@nudt.edu.cn*suteng@huawei.com *

arXiv:2301.04285v1 [cs.DC] 11 Jan 2023

摘要

TAPS 是一种拓扑感知的算子内并行策略搜索算法，通过同时考虑节点内和节点间带宽来生成算子内并行策略。现有的大多数自动并行工作在生成策略时直接将通信量作为通信成本，我们证明这种做法在多节点场景中并非最优。我们设计了一种用于多节点算子内并行策略搜索的拓扑感知成本模型。数值实验表明，*TAPS* 生成的策略最多可减少 *85%* 的通信成本，优于最新的基线方法。

1. 引言

近年来，大规模深度学习（DL）模型因其在 [3, 9, 16],等领域带来的显著性能提升而备受关注，这得益于模型规模和数据集规模的不断扩大。例如，拥有 5400 亿个参数的 PaLM 使用包含 7800 亿个 token 的语料进行训练，这些 token 代表了广泛的自然语言使用场景 [5]。

随着模型大小显著增长，使用单个设备甚至单个节点训练模型已不再实际。因此，研究人员采用分布式深度学习来训练这些模型 [19]。诸如 [17] 之类的手动策略因其出色的性能，已广泛用于训练基于 Transformer 的模型。然而，这种做法往往无法达到最优，因为模型或训练环境发生变化时，最优并行策略也会随之变化；在这种情况下，研究人员和工程师可能需要重新设计策略。

为了免去并行策略设计流程带来的负担，研究人员提出了自动并行算法 [4, 7, 23]，能够针对特定模型和环境找到合适的策略。这些算法首先对并行策略的通信成本进行建模，然后使用动态规划或整数线性规划（ILP）方法来寻找最优策略。

随着模型规模不断增大，单个节点已无法容纳整个大型模型。因此，使用多个节点训练模型变得不可或缺。我们的关键观察是，在多节点环境中，节点内的带宽（节点内带宽）与节点之间的带宽（节点间带宽）并不相同，而且在大多数情况下，节点内带宽远高于节点间带宽。然而，现有的搜索算法直接使用通信量来建模通信成本，忽略了带宽之间的差异，因而得出的策略并非最优。基于这一观察，我们提出了一种名为 TAPS 的拓扑感知并行策略搜索算法。该算法能够捕捉节点内通信与节点间通信之间的差异，从而生成更优的并行策略。

我们首先构建一个拓扑感知成本模型。给定张量的通信轴，该模型能够确定节点间通信次数，并计算拓扑感知通信成本。然后，我们将策略搜索问题形式化为一个整数线性规划问题，并使用第三方求解器求出最终的策略决策。

总而言之，我们的贡献如下：

- 我们证明，在多节点场景下，基于数据量的通信成本模型不足以生成最优的算子内并行策略。
- 我们提出了一种用于优化张量重分布序列的启发式方案。
- 我们分析了多节点环境中的通信，并提出了一种拓扑感知通信成本模型，该模型能够更准确地计算算子并行策略的通信成本。
- 我们设计并实现了 TAPS，这是一种适用于分布式深度学习的策略搜索算法。
- 我们在多种不同配置的模型上对 TAPS 进行了数值评估，并将其与基于体积的搜索进行了比较。实验结果表明，TAPS 找到的策略可将通信成本最多降低 85%。

2. 背景

2.1. 现有并行方法

自 Hinton [8] 于 2012年使用两块 GPU 训练 AlexNet 以来，研究人员提出了许多并行方法，包括数据并行（DP）、模型并行（MP）和流水线并行（PP）。

2.1.1. 数据并行数据并行将数据划分并分发到拥有副本模型的各个设备上。每个设备使用划分后的数据计算梯度，并通过 AllReduce 或广播等通信方式，与其他设备同步梯度或模型参数。因此

*预印本版本，内容可能随时更改。

每次迭代后，所有工作节点上的模型都相同。

2.1.2. 模型并行模型并行将模型参数划分到多个设备上，并让这些设备处理相同的数据。当参数矩阵分别按行和按列划分时，模型并行会产生部分和结果或切片结果。按行模型并行（Row-MP）需要进行同步，以整合不同设备上算子的结果。按列模型并行（Column-MP）只在反向传播时进行同步。

2.1.3. 流水线并行流水线并行将模型中的算子划分为若干阶段，并让设备各自仅持有其中一个或几个阶段。同时，流水线并行将一个小批量数据拆分成若干微批次，并依次将它们送入第一阶段。某个阶段完成计算后，会将结果发送到下一阶段。不同阶段可以同时运行，因此流水线并行形成一条流水线，从而提升性能。

2.2. 算子内与算子间并行

Alpa [23] 将现有并行方法归纳为两类正交类别：算子内并行和算子间并行。算子内并行是指沿某些维度划分算子涉及的张量，将划分后的计算分配给多个设备，并让它们同时执行计算的不同部分。从这个角度看，数据并行可以视为沿批量大小轴划分算子的输入张量和输出张量的一种方案；行模型并行可以视为沿输入通道轴划分算子的输入张量和权重张量的一种方案；列模型并行可以视为沿输出通道轴划分权重张量和输出张量的一种方案。算子间并行（包括流水线并行）则将模型划分为若干包含多个算子的阶段。

本文着重于在多节点环境中生成多维算子内并行策略。

2.3. 策略搜索算法

研究人员提出了自动搜索并行策略的方法。ToFu[21]，、TensorOpt[4]，和 Alpa[23]发现，同一算子的所有不同策略具有相同的计算成本，因此通过最小化计算图的整体通信成本来生成算子内并行策略。ToFu 和 TensorOpt 改进了 OptCNN[7]提出的动态规划算法，以获得更好的结果。Alpa 将搜索问题形式化为整数规划问题，并使用求解器进行求解。然而，这些方法都假设集群各处的带宽相同，忽略了节点内带宽与节点间带宽之间的差异。这一假设可能会限制搜索算法找到最优策略，因为在大规模集群中，节点内带宽远高于节点间带宽。

本文提出了一种拓扑感知通信成本模型，考虑了节点内带宽和节点间带宽，有助于生成更细粒度的策略。

3. 概述

TAPS 是一种通过最小化计算图的通信成本来生成算子内并行策略的算法。TAPS 将计算图 $G=(V,E)$ 和设备图 $D=(V_D,E_D)$ 作为输入，并输出一个分区集合 P，其中包含每个算子$v_i \in V$ 在 G 中的策略决策。计算图包含算子信息，例如形状和算子类型。设备图则指明设备类型以及设备之间的带宽。TAPS 分两步给出解：首先，TAPS 创建一张辅助图，其中每个节点表示一个采用特定策略的算子，并计算辅助图中每条边 (u,v) 的权重，该权重等于v 的算子内通信成本，加上u 与v之间的张量重分布通信成本。随后，TAPS 利用辅助图中的信息，将搜索问题形式化为一个整数线性规划问题，并使用第三方求解器求得最优策略。

4. 通信成本模型

本节详细介绍我们提出的拓扑感知通信成本模型。首先说明基于通信量的成本模型的细节。基于该模型，我们利用通信量和有效带宽计算相应的拓扑感知通信成本。

4.1. 基于通信量的成本模型

以往工作 [4, 18, 20]通过符号化计算各策略的通信量来对其通信成本建模。算子的通信量由算子内通信和算子间通信组成。算子内通信用于归约计算过程中产生的部分和。算子间通信则转换张量，以适配后续算子的策略。

4.1.1. 算子内通信 以矩阵乘法为例，其前向计算如公式所示1，反向计算如公式所示2和公式3。

$$Y = XW \tag{1}$$

$$\delta W = X^T E_y \tag{2}$$

$$E_x = E_y W^T \tag{3}$$

令d、r、c 分别表示矩阵乘法算子的数据并行（DP）[10], 行模型并行和列模型并行 [17]度数；$p = drc$ 表示设备总数，且该总数为 2 的幂。

然后我们将矩阵X和W拆分如下：

$$X = \begin{bmatrix} X_{11} & X_{12} & \dots & X_{1r} \\ X_{21} & X_{22} & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ X_{d1} & X_{d2} & \dots & X_{dr} \end{bmatrix}, \quad W = \begin{bmatrix} W_{11} & W_{12} & \dots & W_{1c} \\ W_{21} & W_{22} & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ W_{r1} & W_{r2} & \dots & W_{rc} \end{bmatrix}.$$

将矩阵X和W切分后，我们将它们的子块分配到相应的设备上。如图 1.(a) 所示，其中每个立方体代表一个设备，X的每个子块都沿c轴进行复制，而W则沿d轴进行复制。如图 1.(b)(c) 所示，随后我们在每个设备上计算Y的局部结果，并通过通信将其汇总以形成最终矩阵Y。该通信是对局部结果执行的归约操作，在数学上等价于公式 4。

$$Y_{ij} = \sum_{k=1}^{r} X_{ik} W_{kj} \tag{4}$$

最终矩阵Y拆分如下：

$$Y = \begin{bmatrix} Y_{11} & Y_{12} & \dots & Y_{1c} \\ Y_{21} & Y_{22} & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ Y_{d1} & Y_{d2} & \dots & Y_{dc} \end{bmatrix},$$

其中，每个子块Y_{ij}沿d轴复制。

假设我们使用带宽最优的环形全归约算法 [15],，则每个设备上对Y的结果进行累加的矩阵乘法算子的通信量（即行模型并行的通信量）为：

$$V_{AR}^{Y} = \frac{2(device_num-1)}{device_num} data_size = \frac{2(r-1)b\ out}{drc}. \tag{5}$$

类似地，通过计算分别对δW和E_X的结果进行累加时的通信量，我们可以得到矩阵乘法算子中数据并行和列模型并行的通信量，分别为：

$$V_{AR}^{\delta W} = \frac{2(d-1)in\ out}{drc}, \tag{6}$$

$$V_{AR}^{E_X} = \frac{2(c-1)b\ in}{drc}. \tag{7}$$

最后，矩阵乘法算子的总通信量为：

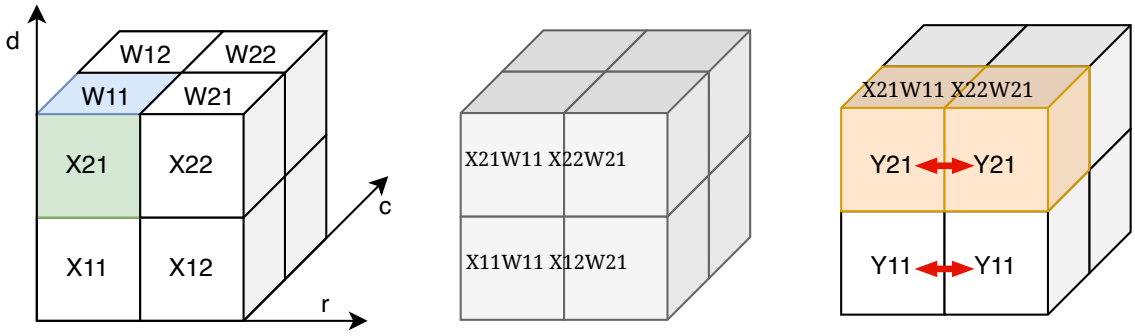
$$Volume = \frac{2((d-1)in\ out + (r-1)b\ out + (c-1)b\ in)}{drc} \tag{8}$$

4.1.2. 算子间通信当两个算子之间发生张量重分布时，就会产生算子间通信。张量重分布是由多个重分布算子组成的序列，例如全收集、切片和全互换。本小节将介绍我们生成合适的重分布算子序列的解决方案。

设O_{out}、O_{in}表示两个算子，T 表示O_{out}的输出 n维张量，同时也是O_{in}的输入张量。$S_T = [s_0, s1, ..., s_{n-1}]$是 T 分区前的形状。假设$O_{out}$和$O_{in}$的设备矩阵深度分别为$h_{out}$和$h_{in}$。$O_{out}$和$O_{in}$中的设备矩阵分别为$\mathscr{D}_{out} = [d_{out,h_{out}-1}, d_{out,h_{out}-2}, ..., d_{out,0}]$和$\mathscr{D}_{in} = [d_{in,h_{in}-1}, d_{in,h_{in}-2}, ..., d_{in,0}]$。$O_{out}$和$O_{in}$中 T 的张量映射分别为$M_{in} = [m_{out,0}, m_{out,1}, m_{out,n-1}]$和$M_{out} = [m_{in,0}, m_{in,1}, ..., m_{in,n-1}]$。要执行张量重分布，$O_{in}$和$O_{out}$的设备矩阵和张量形状必须相同。我们通过两个步骤使它们统一。步骤 1 中，我们对两个设备矩阵中的某些维度进行因式分解，以统一设备矩阵；这可能会导致两个算子中的 T 形状不一致。因此，步骤 2 中还需要在统一后的设备矩阵下统一张量形状。需要注意的是，这两步统一不会改变张量的物理分布。表 1 展示了在O_{out}和O_{in}之间统一一个二维张量的示例。步骤 1 中，我们对两个设备矩阵中的“8”进行因式分解，并分别将其替换为O_{out}和O_{in}的因式分解结果$[4,2]$和$[2,4]$。同时，修改设备矩阵时，我们必须相应地更改张量映射和形状。由于步骤 1 中张量形状发生了变化，因此在推导张量重分布算子之前，需要再次统一张量形状。步骤 2 中，我们对O_{in}和O_{out}中的张量进行重塑，使它们具有相同的形状，并同时修改张量映射。

统一设备矩阵和张量形状后，我们可以推断出重分布算子。一种简单的重分布方法是沿所有工作节点执行全收集，然后沿非重复轴进行切分。为了降低通信成本，我们采用启发式算法1来生成张量重分布算子。我们的算法包含三项优化。首先，我们仅沿张量中必要的轴执行全收集，这些轴在O_{out}中被切分，并在O_{in}中被复制。其次，我们调整重分布序列的顺序，将有依赖关系的切片置于全收集之前，以减少全收集产生的通信量。第三，我们用全互换算子替换隐式置换（即沿设备矩阵中的同一轴执行全收集和切片），从而进一步减少通信量。在算法1中，*InferSlice*找出所有必要的切片操作，并将其追加到算子序列S中。如果没有更多的切片操作，*InferSlice*会将*S_flag*设为*False*。类似地，*InferAll2All*和*InferAllGather*对全交换操作和全收集操作执行相同的操作。表3 展示了一个示例，说明如何运用上述三项优化来微调重分布序列。

最后，我们通过累加序列S中重分布算子的通信量，得到此类张量重分布的算子间通信量。假设我们使用带宽最优环形全收集



(a) 张量 X 和 W 的分布　(b) 本地计算在每个设备上　(c) 沿 r 轴通信以形成张量 Y

图 1：矩阵乘法算子在 8 个设备上的多维 Intra-P，其中 $d=r=c=2$

表 1：张量T的统一设备矩阵和形状

Step	算子	设备矩阵	张量映射	张量形状
0：初始	O_{out}	$[2,8]$	$[1,0]$	$[s_0,s_1]$
	O_{in}	$[8,2]$	$[1,0]$	$[s_0,s_1]$
1：统一设备矩阵	O_{out}	$[2,4,2]$	$[2,1,0]$	$[s_0,4,s_1/4]$
	O_{in}		$[2,1,0]$	$[2,秒_0/2,\ 秒_1]$
2：统一张量形状	O_{out}	$[2,4,2]$	$[2,-1,1,0]$	$[2,s_0/2,\ 4,\ s_1/4]$
	O_{in}		$[2,1,0,-1]$	

表2：$M_{from}=[-1,-1,2,-1,3]$ 与 $M_{to}=[1,-1,-1,0,3]$ 之间的张量重分布

Step	操作	张量映射	通信量
0：初始	全收集（$d_1,1$）	$[-1,-1,2,-1,3]$	$\frac{d_1d_2d_3-1}{d_1d_2d_3}$ 大小(T)
	全收集$(d_2,2)$	$[-1,-1,-1,-1,3]$	
	全收集$(d_3,4)$	$[-1,-1,-1,-1,-1]$	
	切片$(d_1,0)$	$[1,-1,-1,-1,-1]$	
	切片$(d_0,3)$	$[1,-1,-1,0,-1]$	
	切片$(d_3,4)$	$[1,-1,-1,0,3]$	
1：移除	全收集$(d_1,1)$	$[-1,-1,2,-1,3]$	$\frac{d_1d_2-1}{d_1d_2d_3}$ 大小(T)
	全收集$(d_2,2)$	$[-1,-1,-1,-1,3]$	
	切片$(d_1,0)$	$[1,-1,-1,-1,3]$	
	切片$(d_0,3)$	$[1,-1,-1,0,3]$	
2：重排	切片$(d_0,3)$	$[-1,1,2,0,3]$	$\frac{d_1+d_2-2}{d_0d_1d_2d_3}$ 大小(T)
	全收集$(d_1,1)$	$[-1,-1,2,0,3]$	
	切片$(d_1,0)$	$[1,-1,2,0,3]$	
	全收集$(d_2,2)$	$[1,-1,-1,0,3]$	
3：替换	切片$(d_0,3)$	$[-1,1,2,0,3]$	$\frac{d_1d_2-1}{d_0d_21d_2d_3}$ 大小(T)
	全互换$(d_1,1\to 0)$	$[1,-1,2,0,3]$	
	全收集$(d_2,2)$	$[1,-1,-1,0,3]$	

算法；全收集的通信量为：

$$V_{AG} = (device_num - 1)data_size; \tag{9}$$

对于全互换算子，每个设备只需向通信组中的其他设备发送 $1_{\text{设备_数量}}$ 份不同的数据。因此，全互换的通信量为：

$$V_{A2A} = \frac{(device_num - 1)data_size}{device_num}. \tag{10}$$

如表 2所示，算法 1 生成的算子序列的通信量要小得多。

接着，我们将带宽差异纳入基于通信量的成本模型，从而构建拓扑感知成本模型。

算法 1： 优化张量重分布

数据： $\mathscr{D} = [d_{h-1}, d_{h-2}, ..., d_0]$;
$M_{f\text{从}} = [m_{\text{从}}, 0, m_{\text{从}}1, \ldots, m_{\text{从}}, n-1]$;
$M_{to} = [m_{to}, 0, m_{to}, 1, m_{to}, n-1]$
结果： 重分布算子序列S。

当 $M_{from} \neq M_{to}$ 时
  $S_flag \leftarrow$ 真;
  当S_标志 时
    $S_flag \leftarrow Infer_{\text{切片}}(M_{from}, M_{to}, S)$;
    $A2A_flag \leftarrow True$;
    while$A2A$_标志 **do**
      $A2A_f$ 标志 $\leftarrow InferAll2All(M_{from}, M_{to}, S)$;
    end
  end
  $AG_flag \leftarrow Infer_{\text{全收集}}(M_{from}, M_{to}, S)$;
  如果 AG_f 标志 $== False$ **那么**
    $AllGatherFirstUndoneDim(M_{from}, M_{to}, S)$;
结束

4.2. 拓扑感知成本模型

基于基于通信量的成本模型，我们提出了一种拓扑感知成本模型，在计算通信成本时还可以考虑带宽差异。TAPS 使用这种拓扑感知成本模型生成更细粒度的策略。

我们的观察是，在多节点环境中，可以同时进行多个属于不同通信组的节点内通信，而且它们都能充分利用带宽；但对于节点间通信，它们需要共享节点之间的链路，因此会降低每个通信组的有效带宽。图 2.(a) 展示了由 2 个 DGX-V100 节点组成的环境，其中节点内通信使用高带宽 NVLink，节点间通信则使用 100GBps InfiniBand。在图 2.(b) 中，我们沿轴 0 进行通信。共有 8 个通信组，分别为 (GPU0, GPU1)、(GPU2, GPU3) 等。每个通信组都有独立的 NVLink 可用，因此有效带宽等于 NVLink 的带宽。图 2.(c) 中也有 8 个通信组，分别为 (GPU0, GPU8)、(GPU1, GPU9) 等。然而，它们都需要通过唯一的节点间链路（即图中的红线）传输数据。由于它们同时通信，因此需要将带宽除以 8。所以，在这种情况下，有效带宽为 $12.5/8 = 1.5625GB/s$ 。

基于这一观察，TAPS 计算 AllReduce、全收集和全互换算子在一个节点内的节点间通信组数量，以获得每个通信组的有效带宽。TAPS 通过将通信量除以有效带宽来计算通信成本。

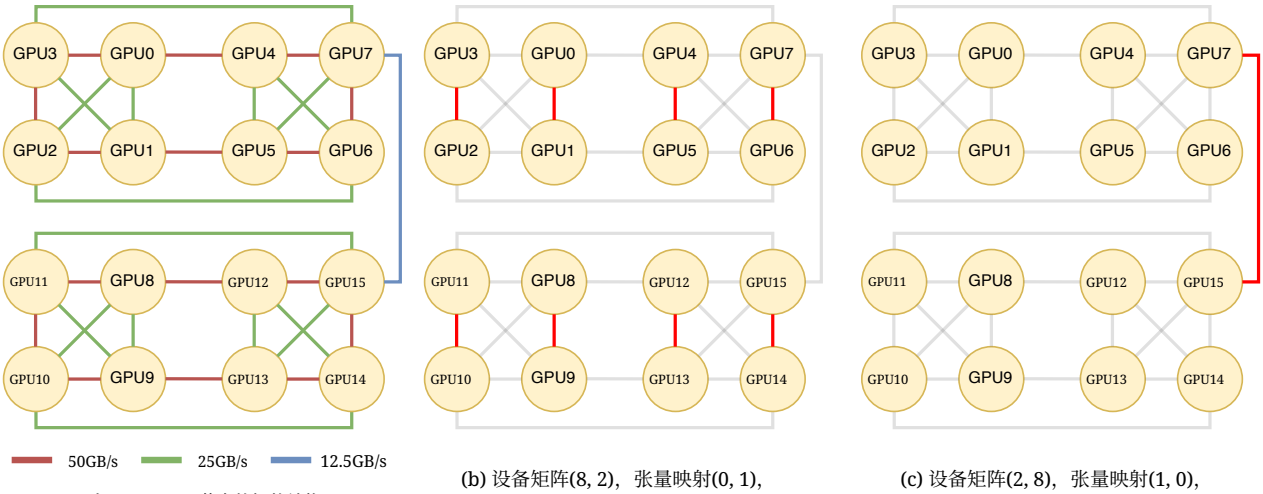
算法2： 推断节点间通信的数量
节点内用于 AllReduce 的通信组

数据： 设备矩阵 $\mathscr{D} = (d_{h-1}, d_{h-2}, \ldots, d_0)$，张量映射 $M = (m_0, m1, \ldots, m_{s-1})$，一个
节点本地设备数量_
结果： 节点间通信组数量ct

剩余_设备 $\leftarrow$本地_设备_数量;
总_设备 $\leftarrow$ 乘积$(\mathscr{D})$;
并行_度数 $\leftarrow$总_设备;
设备_在 $\leftarrow 1$;
对于 $k \leftarrow 0$到 $s-1$ **执行**
  并行_度数 $\leftarrow$ 并行_度数 $\div d_{m_k}$;
对于 $k \leftarrow 0$到 $h-1$ **执行**
  如果 $not\ M.find(k)$ 且 $remain$_设备 > 1 **则**
    如果 $remain$_设备 $> d_k$ **则**
      设备_在 $\leftarrow$ 设备_在 $\times d_k$;
    else
      设备_在 $\leftarrow remain$_设备;
    $remain$_设备 $\leftarrow remain$_设备 $\div d_k$;
结束
如果设备_在 $\geq$ 并行_度数 **则**
  $ct \leftarrow 0$;
否则如果设备_在 > 1 **则**
  $ct \leftarrow$ 本地_设备_数量 $\div$设备_在;
否则
  $ct \leftarrow$本地_设备_数量;

4.2.1. AllReduce对于任意 AllReduce 算子，我们首先使用算法 2 计算其输入张量的节点间通信次数。算法 2以设备矩阵、所通信张量的张量映射以及节点中的设备数量为输入，然后推断需要进行节点间通信的通信组数量。

节点间通信次数表示同时与其他节点进行通信的张量通信组数量。例如，假设我们正在执行一个 Mat-Mul 算子，其策略为 $(d,r,c) = (8,2,2)$，并使用图 3所示的不同设备映射。同色立方体表示同一节点内的设备。在图 3(a) 中，有 4 个不同的 δW



(a) 2 个 DGX-V100 节点的拓扑结构

(b) 设备矩阵(8, 2)，张量映射(0, 1)，通信轴：0

(c) 设备矩阵(2, 8)，张量映射(1, 0)，通信轴：1

图2：2 个 DGX-V100 节点上的节点内通信和节点间通信

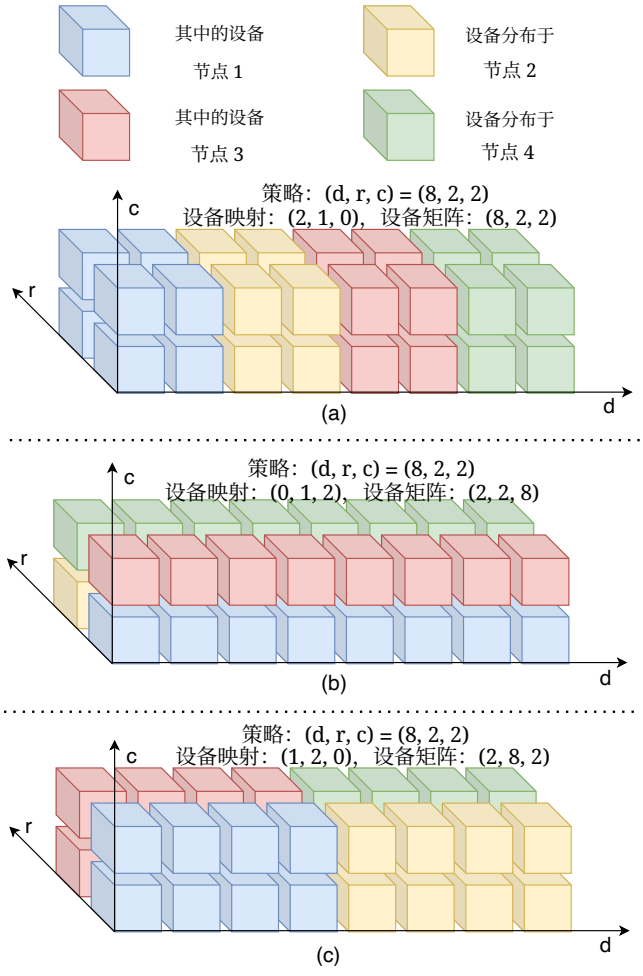


图3：不同设备映射和设备矩阵的设备分布

节点内的分区，且它们都需要与其他节点通信。因此，这种情况下节点间通信次数 $ct_{\delta W}$为 4。图 δW的节点间通信次数$ct_{\delta W}$ 在图 3(a)(b)(c) 中分别为 4、0 和 2。

利用ct的结果，我们可以计算有效带宽B_e:

$$B_e = \begin{cases} B_{intra} & ct_T = 0, \\ B_{inter}/ct_T & ct_T > 0, \end{cases} \tag{11}$$

其中，B_{intra} 是节点内通信带宽，而B_{inter} 是节点间通信带宽。

最后，AllReduce算子的通信成本为:

$$C_{AR} = V_{AR}/B_e. \tag{12}$$

4.2.2. 全收集与AllReduce不同，全收集使用算法3来计算全收集的节点间通信次数。算法3 可以计算全收集在一个设备节点中的重复度r，并推断通信组中有多少设备位于一个节点内。利用这些信息，它随后输出相应全收集的ct 值。TAPS还使用公式11 来计算全收集的B_e 。全收集算子的通信为:

$$C_{AG} = V_{AG}/B_e. \tag{13}$$

4.2.3. 全互换与采用环形拓扑进行通信的 AllReduce 和全收集不同，全互换使用点对点（P2P）通信，在通信组内交换数据。AllReduce 和全收集中，每个节点只有一条发送链路和一条接收链路；而在全互换中，每个节点会建立$p-1$条发送链路和接收链路，与通信组内的其他节点相连，其中p 是全互换通信组中的设备数量。这可能会影响我们用于计算通信成本的通信量。因此，我们需要重新计算全互换的通信

算法3： 推断节点间通信
组数：节点内用于全收集的

数据： 设备矩阵 $\mathscr{D}=(d_{h-1}, d_{h-2}, \ldots, d_0)$，张量映射 $M=(m_0,m1, \ldots, m_{s-1})$，Gather 轴 g，节点内设备数量 本地设备数量。_ _
结果： 节点间通信次数ct

剩余_设备 ←本地_设备_数量;
并行_度数 ←d_{m_g};
临时_设备_数量 ← 1;
重复_次数 ← 1;
对于 $k \leftarrow 0$ 至 $g-1$ **执行**
  临时_设备_数量 ← 临时_设备_数量 $\times d_k$;
  如果 *not M. find*(k) **则**
    重复_数字 ← 重复_数字 $\times$天$_k$;
结束
如果*repeat_num* $>$*local*_设备_*num* **则**
  repeat_num ← *local*_设备_*num*;
如果*temp*_设备_*num* $\geq$*local*_设备_*num* **则**
  ct ← 本地_设备_数量 ÷重复_数量;
否则
  剩余设备 ←_
    本地_设备_数量 ÷临时_设备_数量;
  如果 剩余_设备 $\geq$ 并行_度 **则**
    ct ← 0;
  else
    ct ← 临时_设备_数量 ÷重复_数量;

量。假设 p 台设备中有 k 台位于同一节点内，且张量大小为 T_S。那么，对于节点内的任意设备，存在 $k-1$ 次节点内通信，通信量为 T_S/p；以及 $p-k$ 次节点间通信，通信量为 T_S/p。这 k 台设备与其他节点建立的连接总数为 $(p-k)k$，每条连接传输 T_s/p 的数据量。因此，对于该全互换通信组，通过节点间链路传输的节点间通信量为 $k(p-k)T_S/p$。假设一个节点内有 l 台设备，则该节点内有 l/k 个不同的全互换通信组。此外，假设这些通信组的重复度为 r。重复的张量切片可以通过利用高带宽链路在节点内同步，共享相同的通信结果。于是，在一个节点内，$ct=l/(kr)$ 个组会同时使用节点间带宽进行通信，除非 p 等于 k。k 和 r 的值也可以通过算法3推导得出。有效带宽 B_e 根据公式11计算。因此，全互换的通信成本为：

$$C_{A2A}=\begin{cases}\frac{V_{A2A}}{B_e} & p=k,\\ \frac{k(p-k)}{p-1}\frac{V_{A2A}}{B_e} & p>k,\end{cases} \tag{14}$$

5. 辅助图

辅助图 $G_A=(V_A, E_A)$ 是计算图 G 的扩展，其中每个节点 $v_a \in V_A$ 都表示一个唯一的

算法4： GenerateAuxiliaryGraph

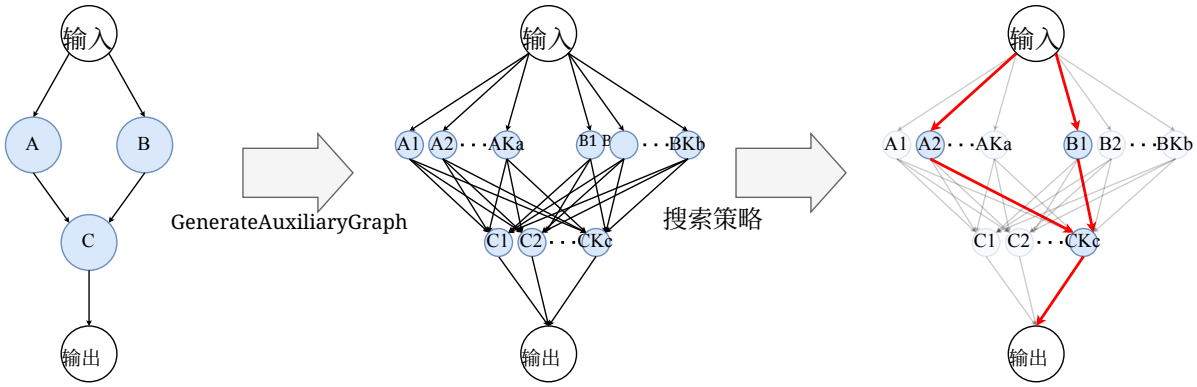
数据： 计算图 $G=(V, E)$、设备图 $D=(V_D, E_D)$
结果： 辅助图 $G_A=(V_A, E_A)$

$V_A \leftarrow /0, E_A \leftarrow /0$;
设备_*num* ← $|V_D|$;
对于 $(u,w) \in E$ **执行**
  U_A = GenerateStrategySet(u,设备_*num*);
  W_A =GenerateStrategySet(w, 设备_*num*);
  $V_A = V_A \cup U_A \cup W_A$;
  对于$u_a \in U_A$ **执行**
    对于$w_a \in W_A$ **执行**
      $E_A = E_A \cup \{(u_a, w_a)\}$
    end
  end
结束

其原始顶点 $v \in V$ 的策略。我们使用算法4生成辅助图，如图4(a)(b)所示。对于每个 $v_a \in V_A$，我们用其原始算子和一个唯一的策略对其进行标记。算法4中的函数 GenerateStrategySet 会枚举输入算子的所有可能策略，并为其创建相应的辅助节点。更具体地说，当算子中有 p 个可分区轴，且该算子由 $N=2^n$个设备承载时，GenerateStrategySet 将生成 $\sum_{i=1}^{\min(p,n)} i!\binom{p}{i}\binom{n-1}{i-1}$ 种不同的并行策略。算子的并行策略由并行度以及各轴的映射构成。我们可以利用并行度确定算子中每个相关张量的分区，并根据映射将它们放置到相应的设备上。例如，假设有一个执行计算 $Y=XW$ 的矩阵乘法（MatMul）算子，它可以沿三个轴进行分区：b轴、输入轴和输出轴；X、W 和 Y 未分区时的形状分别为 (b,in)、(in, out) 和 (b,out)。表 3 展示了以该矩阵乘法算子和设备数量 4 作为输入时，GenerateStrategySet 生成的策略集。以表 3 中的 u_2 in 为例，输出轴上的数字 2 表示张量 W 和 Y 沿输出轴切分为 2 个部分；设备映射 $(-1,1,0)$ 表示 b轴、输入轴和输出轴的映射值分别为 -1、1 和 0。这里的 -1 表示张量沿 b轴复制。这里的 0 表示输出轴在集群中被划分到最内层，这样在通信时可能具有较高的带宽。设备矩阵 (1, 2,2) 根据各轴的并行度和设备映射计算得出，它是设备的分层逻辑拓扑。

创建辅助图的顶点和边后，我们使用拓扑感知成本模型计算所有$C_{u_a w_a}$ 的通信成本$e_a=(u_a,w_a) \in E_A$。边e_a 的权重等于w_a 的算子内成本加上u_a 与w_a之间的算子间成本。

然后，我们可以通过选择顶点和



(a) 原始计算图　　(b) 辅助计算图　　(c) 策略决策示例

图4：辅助计算图

辅助图中的边。图 4展示了搜索结果的一个示例，其中蓝色顶点表示选中的策略。

6. 基于整数线性规划的搜索策略

我们将策略搜索问题形式化为如下所示的整数线性规划问题：

$$\min \quad \sum_{(i,j)\in E_A} B_{ij}C_{ij} \tag{15}$$

$$\text{s.t.} \sum_{v_a\in V_A} X_{v_a} = 1, \qquad \forall v \in V \tag{16}$$

$$\sum_{(i,v_a)\in E_A} B_{iv_a} = X_{v_a} \times in_degree(v)$$

$$\sum_{(v_a,k)\in E_A} B_{v_ak} = X_{v_a} \times out_degree(v), \forall v_a \in V_A \tag{17}$$

$$\sum_{(i,j)\in E_A} B_{ij}M_{ij} < Device_Memory, \tag{18}$$

$$B_{ij}, X_k \in \{0,1\}, \qquad \forall (i,j) \in E_A, \forall k \in V_A \tag{19}$$

其中，X_{v_a}、B_{ij} 是待求解的布尔值，分别表示是否选择顶点 $v_a \in V_A$ 以及边 $(i,j) \in E_A$。C_{ij} 和 M_{ij} 分别是边 $(i,j) \in E_A$ 的通信成本和内存成本。公式 16 告知求解器，对于所有 $v \in V$，只选择一种策略。公式 17 限制任意 $v_a \in V_A$ 的入度和出度与其原始顶点 $v \in V$ 相同。为避免为 $v \in V$ 选择多种策略，如果未选择顶点 $v_a \in V_A$，则将其入度和出度设为零。公式 18 限制求解器生成的整体策略不得超出设备内存。

我们没有采用动态规划，而是出于两个原因使用整数线性规划。首先，像 [7, 21] 这样的动态规划方法无法在处理过程中纳入整体内存开销，因此可能会生成超出内存约束的策略。尽管像 [4] 这样的方法通过设置通信与内存成本界限来避免这一缺陷，但在为大规模模型生成策略时，其计算复杂度难以接受。其次，我们可以直接使用高性能第三方求解器来求解整数线性规划问题，从而节省优化求解器运行时间所需的精力。

7. 评估

我们通过比较基于体积的搜索和基于拓扑感知的搜索所生成策略的通信成本来评估 TAPS。在评估中，我们假设节点内带宽为 60GB/s，节点间带宽为 6GB/s。这两个数值是我们使用 nccl-tests[1] 在两台 8-V100 节点上测试得到的峰值带宽。此外，我们假设所有通信都能充分利用带宽，并且运行环境为同构环境。

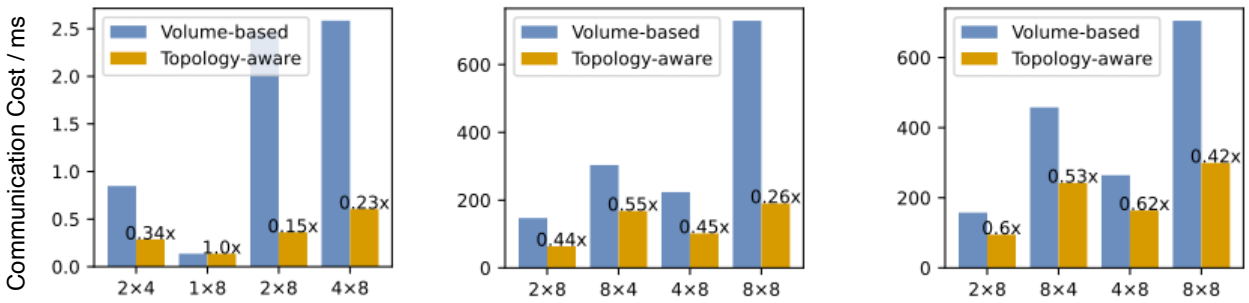
7.1. 搜索运行时间

我们测试了 AlexNet[8] 和 Megatron-LM[14, 17]的策略搜索运行时间。请注意，基于 Transformer 的网络主体由多个结构相同的层组成。当所用设备同构时，相同结构始终采用相同策略，因此我们只为网络中的一个 Transformer 层搜索策略。使用一台 16 核 3.2GHz Intel i9-12900K CPU，所应用的求解器可在几秒内求出策略。在搜索运行时间实验中，我们假设每个节点有 8 个设备。

表4展示了求解策略运行时间的一些示例，其中V_D表示设备总数，而 $|E_A|$ 表示辅助边的数量。搜索时间与模型的参数数量无关，而与模型的算子数量及设备总数有关。例如，Megatron-LM 1.7B 和 3.6B 的 Transformer 层结构相同，但参数数量不同。我们按照以下配置，在总计 32 台设备上分别为 1.7B 和 3.6B 搜索算子内策略。如表 [14],所示，4 ， $|E_A|$ 及其耗时的数量级相同。

表 3：由 GenerateStrategySet 生成的、在 4 台设备上进行分区的矩阵乘法算子（$Y = XW$）的辅助节点

Node	b 轴	in 轴	out 轴	X 形状	W 形状	Y 形	设备映射	设备矩阵
u_1	1	1	4	(b，输入)	(输入，输出/4)	(b，输出/4)	(-1, -1, 0)	(1, 1, 4)
u_2	1	2	2	(b, in/2)	(输入/2，输出/2)	(b, out/2)	(-1, 1, 0)	(1, 2, 2)
u_3	1	2	2	(b, in/2)	(输入/2，输出/2)	(b，out/2)	(-1, 0, 1)	(1, 2, 2)
u_4	1	4	1	(b, in/4)	(输入，输出/4)	(b，输出)	(-1, 0, -1)	(1, 4, 1)
u_5	2	1	2	(b/2，位于)	(输入，输出/2)	(b/2，输出/2)	(-1, 1, 0)	(2, 1, 2)
u_6	2	1	2	(b/2，输入)	(输入，输出/2)	(b/2, out/2)	(-1, 0, 1)	(2, 1, 2)
u_7	2	2	1	(b/2, 内/2)	(in，out)	(b/2, 输出)	(1, 0, -1)	(2, 2, 1)
u_8	2	2	1	(b/2, 内/2)	(输入/2，输出)	(b/2，输出)	(0, 1, -1)	(2, 2, 1)
u_9	4	1	1	(b/4，输入)	(输入，输出)	(b/4，输出)	(0, -1, -1)	(4, 1, 1)



(a) AlexNet (b) Megatron-LM 1.7B (c) Megatron-LM 3.6B

图5：不同模型上的拓扑感知搜索与基于体积的搜索对比

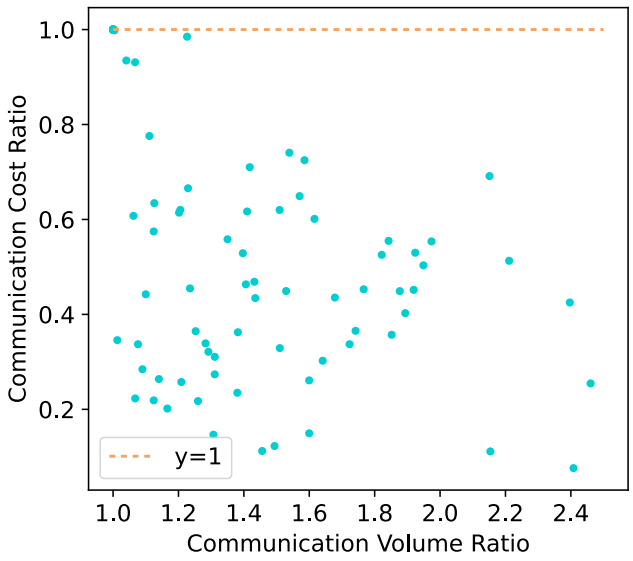


图6：拓扑感知搜索与基于体积的搜索的比率

表 4：求解器的策略搜索时间

模型	$\|V_D\|$	$\|E_A\|$	Time
AlexNet	8	$> 3 \times 10^3$	< 0.1秒
AlexNet	16	$> 10^4$	< 0.2秒
AlexNet	64	$> 5 \times 10^4$	< 0.8秒
Megatron-LM 1.7B	32	$> 3 \times 10^4$	< 0.4秒
Megatron-LM 3.6B	32	$> 3 \times 10^4$	< 0.4秒
Megatron-LM 1T	8	$> 4 \times 10^3$	< 0.1秒

7.2. 与基于体积的搜索进行比较

我们比较基于体积的搜索和拓扑感知搜索所求得策略的通信成本。在实验中，我们针对卷积网络 AlexNet 和基于 Transformer 的网络 Megatron-LM 搜索策略。进行基于体积的搜索时，我们将辅助边的通信成本替换为相应的通信量，然后使用相同的求解器搜索策略。接着，我们使用拓扑感知成本模型计算基于体积的搜索所得结果的通信成本。比较结果为

如图5所示，蓝色柱表示基于体积的搜索结果的通信成本，橙色柱表示拓扑感知搜索结果的通信成本。我们以 AlexNet、Megatron-LM 1.7B 和 3.6B 为例。如5(a) 所示，当只有一个节点时，两种不同搜索结果的通信成本相同。这很容易理解，因为在这种情况下不存在节点间通信。我们的实验表明，TAPS 总能找到优于基于体积的搜索所找到策略的策略。对于在两个 8设备节点上为 AlexNet 搜索策略的情况，通信成本甚至降低了 85%。此外，我们将所有实验结果汇总于图6中，其中每个点代表在特定设备拓扑下对一个模型进行的一次搜索。 x轴表示拓扑感知通信量与基于体积的通信成本之比；y轴表示拓扑感知通信成本与基于体积的通信成本之比。如图所示，图中的所有点都位于直线$y = 1$上或其下方，这意味着拓扑感知搜索总能找到通信成本低于基于体积的搜索的策略。此外，在大多数情况下，拓扑感知搜索可将通信成本降低 20% 以上。

8. 相关工作与讨论

流水线并行。 Alpa、Chimera[12] 和 PipeDream[13] 等自动并行方法能够生成流水线并行策略，从而平衡不同设备上的各阶段。TAPS 的搜索空间与流水线并行正交。因此，我们可以使用 TAPS 为流水线的每个阶段搜索算子内并行策略。

多维张量并行。 Colossal-AI[11] 中的 2D-TP[22], 3 D- TP[2] 以启发式方式生成算子内策略。TAPS 目前不支持 2D-TP，因为 2D-TP 使用广播和归约完成通信，而我们使用全收集、全互换和切片。TAPS 自然包含 3D-TP 的策略，因为

通信与计算重叠。 在我们的实现中，我们假设通信无法与计算重叠，因此可以忽略计算开销。然而，在实际训练中，研究人员[6]会安排计算与通信重叠。我们很难确定重叠度。一种折中方案是手动设置不同维度通信的重叠度。例如，在某些情况下，数据并行的通信可以完全重叠，此时我们可以将重叠度设为 1。

使用回归模型估算成本。 尽管我们假设带宽能够得到充分利用，但我们注意到，当传输数据量较小时，有效带宽非常低。这是因为通信过程中会产生一些开销，例如创建连接和计算平均值。使用回归模型模拟有效带宽的变化，是进一步改进 TAPS 的好方法。

9. 结论

我们提出了 TAPS，这是一种面向多节点环境、拓扑感知的算子内并行策略搜索算法，可生成细粒度的算子内策略。TAPS 能够以启发式方式生成通信成本更低的张量重分布操作。TAPS 根据通信量和有效带宽计算每种策略的通信成本，因此，与仅考虑通信量的方法相比，它能生成更合理的策略。基于通信成本，TAPS 通过构建并利用辅助图，将搜索问题形式化为一个整数线性规划问题，并在几秒内求解。与基于通信量的搜索算法相比，对于多节点环境中的案例，TAPS 生成的策略最多可减少 85% 的通信成本。TAPS 的源代码将公开发布。

参考文献

[1] 访问日期：2022年10月20日。NVIDIA nccl-tests。https://github.com/NVIDIA/nccl-tests。[2] Zhengda Bian、Qifan Xu、Boxiang Wang 和 Yang You。最大化超大规模神经网络分布式训练中的并行度。2021年5月。arXiv:2105.14450。[3] Tom B. Brown、Benjamin Mann、Nick Ryder、Melanie Subbiah、Jared Kaplan、Prafulla Dhariwal、Arvind Neelakantan、Pranav Shyam、Girish Sastry、Amanda Askell、Sandhini Agarwal、Ariel Herbert-Voss、Gretchen Krueger、Tom Henighan、Rewon Child、Aditya Ramesh、Daniel M. Ziegler、Jeffrey Wu、Clemens Winter、Christopher Hesse、Mark Chen、Eric Sigler、Mateusz Litwin、Scott Gray、Benjamin Chess、Jack Clark、Christopher Berner、Sam McCandlish、Alec Radford、Ilya Sutskever 和 Dario Amodei。语言模型是少样本学习器。2020年5月。arXiv:2005.14165。[4] Zhenkun Cai、Xiao Yan、Kaihao Ma、Yidi Wu、Yuzhen Huang、James Cheng、Teng Su 和 Fan Yu。Tensoropt：探索利用自动并行进行分布式深度神经网络训练时的权衡。*IEEE*并行与分布式系统汇刊，33(8):1967–1981，2022。[5] Aakanksha Chowdhery、Sharan Narang、Jacob Devlin、Maarten Bosma、Gaurav Mishra、Adam Roberts、Paul Barham、Hyung Won Chung、Charles Sutton、Sebastian Gehrmann、Parker Schuh、Kensen Shi、Sasha Tsvyashchenko、Joshua Maynez、Abhishek Rao、Parker Barnes、Yi Tay、Noam Shazeer、Vinodkumar Prabhakaran、Emily Reif、Nan Du、Ben Hutchinson、Reiner Pope、James Bradbury、Jacob Austin、Michael Isard、Guy Gur-Ari、Pengcheng Yin、Toju Duke、Anselm Levskaya、Sanjay Ghemawat、Sunipa Dev、Henryk Michalewski、Xavier Garcia、Vedant Misra、Kevin Robinson、Liam Fedus、Denny Zhou、Daphne I ppolito、David Luan、Hyeontaek Lim、Barret Zoph、Alexander Spiridonov、Ryan Sepassi、David Dohan、Shivani Agrawal、Mark Omernick、Andrew M. Dai、Thanumalayan Sankaranarayana Pillai、Marie Pellat、Aitor Lewkowycz、Erica Moreira、Rewon Child、Oleksandr Polozov、Katherine Lee、Zongwei Zhou、Xuezhi Wang、Brennan Saeta、Mark Diaz、Orhan Firat、Michele Catasta、Jason Wei、Kathy Meier-Hellstern、Douglas Eck、Jeff Dean、Slav Petrov 和 Noah Fiedel。PaLM：利用路径扩展语言建模规模，2022，arXiv:2204.02311。[6] Abhinav Jangda、Jun Huang、Guodong Liu、Amir Hossein Nodehi Sa- bet、Saeed Maleki、Youshan Miao、Madanlal Musuvathi、Todd Mytkow- icz 和 Olli Sarikivi。打破分布式机器学习工作负载中计算与通信的抽象壁垒。发表于*ASP- LOS 2022*，2021年5月。[7] Zhihao Jia、Sina Lin、Charles R Qi 和 Alex Aiken。探索隐藏维度以加速卷积神经网络。发表于国际机器学习会议，第2274–2283页。PMLR。[8] Alex Krizhevsky、Ilya Sutskever 和 Geoffrey E Hinton。使用深度卷积神经网络进行ImageNet分类。《*ACM*通讯》，60(6):84–90，2017年6月。[9] Yann LeCun、Yoshua Bengio 和 Geoffrey Hinton。深度学习。*Nature*，521(7553):436–444，2015。

[10] Shen Li、Yanli Zhao、Rohan Varma、Omkar Salpekar、Pieter Noordhuis、Teng Li、Adam Paszke、Jeff Smith、Brian Vaughan、Pritam Damania 和 Soumith Chintala。PyTorch 分布式：加速数据并行训练的实践经验。2020年6月。arXiv:2006.15704。[11] Shenggui Li、Jiarui Fang、Zhengda Bian、Hongxin Liu、Yuliang Liu、Haichen Huang、Boxiang Wang 和 Yang You。Colossal-AI：面向大规模并行训练的统一深度学习系统。2021年10月。arXiv:2110.14883。[12] Shigang Li 和 Torsten Hoefler。Chimera：通过双向流水线高效训练大规模神经网络。发表于国际高性能计算、网络、存储与分析会议论文集，SC '21，美国纽约州纽约市，2021年。计算机协会。[13] Deepak Narayanan、Aaron Harlap、Amar Phanishayee、Vivek Seshadri、Nikhil R Devanur、Gregory R Ganger、Phillip B Gibbons 和 Matei Zaharia。PipeDream：用于深度神经网络训练的通用流水线并行。发表于第*27*届*ACM*操作系统原理研讨会论文集，第1–15页。[14] Deepak Narayanan、Mohammad Shoeybi、Jared Casper、Patrick LeGresley、Mostofa Patwary、Vijay Anand Korthikanti、Dmitri Vain- brand、Prethvi Kashinkunti、Julie Bernauer、Bryan Catanzaro、Amar Phanishayee 和 Matei Zaharia。在GPU集群上高效进行大规模语言模型训练。2021年4月。arXiv:2104.04473。[15] Pitch Patarasuk 和 Xin Yuan。工作站集群的带宽最优全归约算法。《并行与分布式计算杂志》，69(2):117–124，2009年。[16] Andrew W. Senior、Richard Evans、John Jumper、James Kirkpatrick、Laurent Sifre、Tim Green、Chongli Qin、Augustin Žídek、Alexander W. R. Nelson、Alex Bridgland、Hugo Penedones、Stig Petersen、Karen Simonyan、Steve Crossan、Pushmeet Kohli、David T. Jones、David Silver、Koray Kavukcuoglu 和 Demis Hassabis。利用深度学习势能改进蛋白质结构预测。*Nature*，577:706–710，2020年。[17] Mohammad Shoeybi、Mostofa Patwary、Raul Puri、Patrick LeGresley、Jared Casper 和 Bryan Catanzaro。Megatron-LM：使用模型并行训练数十亿参数的语言模型。2019年9月。arXiv:1909.08053。[18] Linghao Song、Fan Chen、Youwei Zhuo、Xuehai Qian、Hai Li 和 Yiran Chen。Accpar：面向异构深度学习加速器的张量划分。发表于*2020*年*IEEE*国际高性能计算机体系结构研讨会（*HPCA*），第342–355页。IEEE。[19] Joost Verbraeken、Matthijs Wolting、Jonathan Katzy、Jeroen Kloppen- burg、Tim Verbelen 和 Jan S Rellermeyer。分布式机器学习综述。《*ACM*计算调查》（*CSUR*），53(2):1–33，2020年。[20] Haoran Wang、Chong Li、Thibaut Tachon、Hongxing Wang、Sheng Yang、Sébastien Limet 和 Sophie Robert。用于并行化的大型深度神经网络高效、系统化划分。载于Leonel Sousa、Nuno Roma 和 Pedro Tomás 编，《 style id='24'>Euro-Par 2021：并行处理，第201–216页，尚，2021年。Springer国际出版公司。[21] Minjie Wang、Chien-chin Huang 和 Jinyang Li。通过自动数据流图划分支持超大型模型。发表于第十四届*EuroSys 2019*会议论文集，第1–17页，2019年。[22] Qifan Xu、Shenggui Li、Chaoyu Gong 和 Yang You。一种用于训练超大型深度学习模型的高效二维方法。2021年4月。arXiv:2104.05343。[23] Lianmin Zheng、Zhuohan Li、Hao Zhang、Yonghao Zhuang、Zhifeng Chen、Yanping Huang、Yida Wang、Yuanzhong Xu、Danyang Zhuo、Joseph E. Gonzalez、Ion Stoica 和 Eric P. Xing。Alpa：为分布式深度学习实现算子间与算子内并行自动化。2022年1月。arXiv:2201.12023。