arXiv:2302.08141v1 [cs.DC] 16 Feb 2023

使用 Rhino 自动并行化大模型：生产级人工智能平台上的系统化方法

张世伟[1,2,*] 刁岚松[1,*] 王思宇[1] 曹宗岩[1] 顾一亮[1]司畅[1] Ziji Shi[1,3] Zhen Zheng[1] 吴川[2] 林伟[1]

[1]阿里巴巴集团 [2]香港大学 [3]新加坡国立大学

摘要

我们介绍 Rhino，这是一套用于在生产环境中的人工智能平台上通过自动并行化加速张量程序的系统。它能将为单个设备编写的张量程序转换为等效的分布式程序，无需用户配置即可扩展至数千台设备。首先，Rhino 基于张量程序在语义上独立的中间表示运行，从而便于将其推广到前所未有的应用场景。此外，它还实现了面向任务的控制器和分布式运行时，以实现最佳性能。Rhino 探索了一个完整、系统化的并行化策略空间，涵盖深度学习（DL）中常用的所有范式，还包括针对非线性模型的步长分区和流水线并行。为高效搜索近乎最优的并行执行计划，我们对生产集群的分析揭示了可加速策略搜索的通用启发式方法。在此基础上，我们设计了两个优化级别，使用户能够灵活权衡搜索时间与策略质量。实验表明，Rhino 不仅能够重新发现经典、研究型和生产级 DL 模型中由专家精心设计的策略，还能为新型模型找出超越现有系统的全新并行化策略。

1 引言

拥有数千亿参数的大型模型展现出前所未有的性能，并推动了新型深度神经网络（DNNs）的研究。例如，用于语言建模的 PaLM [5]拥有 540B 个参数，而用于多模态预训练的 M6 [25]包含 100B 个参数。在商用硬件（例如 GPU）上训练如此大规模的模型，需要精心设计的分布式训练策略，这些策略通常采用多种并行范式，包括数据并行、模型并行和流水线并行。

许多系统都探索了大型模型并行化的自动化。Tofu [45] 使用动态规划算法寻找混合数据并行与模型并行策略。Alpa [49]使用整数线性规划（ILP）实现算子内并行，并使用动态规划实现算子间并行。Unity [44] 通过在并行计算图上进行随机图替换，支持代数变换与并行化的联合优化。这些系统表明，全自动算法能够达到与专家设计策略相当或更优的分布式训练性能。

然而，最先进的模型，尤其是那些需要在大型集群上进行分布式训练的大型模型，仍然倾向于采用人工设计的并行化策略。MT-NLG [42] 使用人工设计的三维并行（混合数据并行、模型并行和流水线并行），在 DeepSpeed [34] 和 Megatron [29]上进行训练。GLaM [7] 使用 GSPMD [47]，通过人工设计的单程序多数据并行（混合数据并行和模型并行）进行训练。这促使我们思考：当前的自动并行化框架存在哪些障碍，使其无法应用于生产环境?

我们的观点可归纳为以下四个方面。

首先，现有的自动并行化系统不够通用，因为它们通常基于深度神经网络模型的语义来设计分布式训练策略，例如“层”的概念、张量轴的含义（“批量大小”、“词元”或“隐藏层宽度”等）。例如，AccPar [43]基于张量维度的语义，考虑了三种并行性。PipeDream [28] 以“层”为单位对模型进行划分。Whale [17] 利用对层组的手动标注来实现流水线并行。然而，随着新的深度神经网络架构不断被提出，这些定制层的概念可能会过时，并阻碍新型并行策略的实现。例如，Tofu [45] 利用深度神经网络框架提供的高级编程接口来检测并合并循环神经网络层中展开的时间步；对于不包含循环神经网络层的新模型（如基于 Transformer 的模型），该方法将会失效。

*Shiwei 和 Lansong 贡献相同。

此外，尽管将分布式训练策略与模型语义绑定有助于推理和调试，但这会限制策略空间，并可能错失重要的优化机会。例如，在 PyTorch 中，层通常映射为一个 nn.Module 对象，是由多个算子组成的语义组，而这些算子通常是根据代码复用需求选取的。使用层作为划分单位，会使编程约定与模型训练性能产生不必要的耦合。

其次，许多自动并行化系统支持的并行范式范围有限，未能提供完整且系统化的并行化方法空间。例如，PipeDream [28] 不支持张量并行，而 Tofu [45] 未考虑流水线并行。此外，目前没有任何现有的 SPMD 系统支持步长分区，相关内容将在第 4.1 节中介绍。随着模型规模不断增大，结合所有已知并行范式以最大限度提高并行化程度已变得至关重要。

第三，现有自动并行化系统所使用的启发式方法大多基于直觉，且仅在非常有限的场景中进行了评估。由于并行化策略的搜索空间呈指数级增长，所有现有自动并行化系统都采用启发式方法来缩小搜索空间并加快搜索速度。这些启发式方法会显著影响生成的并行化策略的质量，却主要凭直觉构思而来，且很少通过多样化的工作负载验证其稳健性。

最后，许多自动并行化系统近乎黑箱，且无法干预。例如，如果自动并行化系统输出的策略超出 GPU 内存限制并导致内存溢出（OOM）错误，用户可能不清楚该如何解决这一问题。因此，可靠的自动并行化系统应提供充分的控制能力，以应对意外情况。

为解决这些问题，我们提出了 Rhino，这是一种全自动并行训练系统，其设计遵循以下指导原则：

▷ Rhino 在 HLO [10] 层级运行；HLO 是 XLA 使用的编译器中间表示（IR） [46]。HLO IR 包含完整的计算图，包括前向传播和反向传播，从而支持整体优化，并允许同一层在前向传播和反向传播中采用不同的分片策略。此外，Rhino 不依赖层的概念或张量轴的作用，而只关注操作序列的算术特性，因此，只要程序产生相同的计算结果，就能将模型声明代码（例如，不同的前端库，如 TensorFlow 或 JAX，以及不同的层边界等）与训练性能解耦。此外，Rhino 还支持反向传播以外的新训练方法，例如前向-前向算法 [14]。

▷ Rhino 包含一个完整且系统化的策略空间。除了分片维度，Rhino 还考虑张量分片的步长布局，由此揭示了以往未曾报告过的实用分片策略。我们证明，许多常见的并行化策略，例如 Megatron 的 [29] 模型并行、 ZeRO 优化器 [33]，以及 GShard 的 [22] 混合专家分片策略，都是 Rhino 所探索策略的特例。为了高效探索庞大的搜索空间，我们研究了生产环境深度神经网络模型中的常见模式，并提出了一种结合整数线性规划（ILP）和动态规划（DP）的搜索算法。

▷ Rhino 使用源自生产系统、经数据驱动且经过系统评估的启发式方法，以加快策略搜索。借助在线收集的充分集群任务信息，我们构建了有关系统化优化能力的精确且细致的统计数据，从而改进 Rhino 并支持其在生产环境中的部署。鉴于在 HLO 层级的指数级庞大策略空间中进行穷举搜索并不可行，Rhino 融合了通过分析生产模型所积累的领域知识与洞见，以实现快速且可靠的策略搜索。

▷ Rhino 支持完全自动并行化，同时也允许适当的用户干预：用户可以通过注释以半自动方式调整分区策略。此外，Rhino 提供两个优化级别，以便在不同场景下权衡搜索时间与策略质量。最后，我们的实践增强了对边界情况的处理能力，从而进一步改进 Rhino。

在实现方面，Rhino 构建任务图并采用静态调度，通过减少将在第 6 节中说明的协作开销来实现更高的吞吐量。我们的实验表明，与最先进的系统相比，Rhino 的吞吐量可提升 19% ~25%。对于大型模型，Rhino 包含一种激进的优化级别，可将策略搜索加速 1。5× ~3×，且仅带来 10% 的性能开销。基于这些对生产友好的优势，Rhino 支持在生产级人工智能平台上进行多模型训练。

2 背景

2.1 分布式深度神经网络训练中的常见模式

单程序多数据并行。 单程序多数据并行 [47, 49] 将数据并行和模型并行进行了泛化，使每种并行方式都可以沿其任意维度进行均匀分区，并分布到不同设备上。当两个算子的分区方法不兼容，即它们生成和消费的张量采用不同的分片方法时，MPI 风格的 [27]集合通信便是必要的，用于拆分和合并张量分片。常见的集合通信算子包括 全归约、全收集、归约散播和 全互换。

流水线并行。 采用流水线并行时， [8, 15, 28],模型被划分为一系列阶段，并分布到不同的设备组上。输入数据批次被划分为微批次，这些微批次的执行以流水线方式进行编排。激活值及其梯度在各阶段之间通过点对点通信进行传输。阶段划分和调度至关重要，因为它们会直接影响流水线并行的性能，而且要实现最优决策并非易事。因此，理想的流水线策略应在平衡各阶段计算时间的同时，尽量减少阶段之间的传输。

研究人员一直在探索将 SPMD 并行与流水线并行相结合，用于训练大型模型 [34, 44, 49]。然而，手动设计组合策略需要丰富的专业知识，而且难以推广；与此同时，在大型模型和复杂集群的背景下，自动化搜索必须应对庞大的探索空间，并且大型模型的编译时间长达数小时。因此，要实现高效且可泛化的并行化策略搜索，一种很有前景的方法是利用编码专家知识的启发式方法来引导自动化搜索。

2.2 计算图与 HLO

反向传播是深度学习中占主导地位的训练方法。人工智能包含两个组成部分：计算训练损失的前向步骤，以及计算模型参数梯度的反向步骤。这两个步骤都可以表示为张量程序，即由算子构成、以张量作为输入和输出的序列。许多流行的深度学习框架 [1, 30] 将张量程序编码为有向无环图（DAG），称为计算图。在计算图中，每条边都表示一个张量，而顶点表示对这些张量执行的操作。

不同的深度学习框架拥有各自不同的算子集合，并且可能针对不同硬件（例如 TPU 或 GPU）进行优化。为避免在支持各种前端平台和后端硬件时重复投入工程资源，人工智能编译器（例如 XLA [46] 和 TVM [4]）通常会为张量程序引入与平台和硬件无关的中间表示（中间表示）。人工智能编译器会将使用不同前端平台定义的模型转换为由中间表示表示的统一计算图，随后执行一系列优化过程。每个过程都会应用特定的优化（例如死代码消除），并将给定的中间表示转换为优化后的中间表示。然后，优化后的中间表示会被降低为低级中间表示，以进行硬件特定优化和编译。

2.3 近期模型

我们介绍了近期提出的三种模型，它们不仅规模庞大，架构也错综复杂，凸显了分布式训练所带来的复杂性。通过严格考察训练过程，我们评估了相应的挑战，并探讨了训练这些模型时可能存在的机遇。

混合专家（MoE）模型。MoE 模型在自然语言建模领域取得了显著成功[7, 9, 40]。MoE 模型由包含条件激活专家的稀疏层（MoE 层）组成。路由层决定哪些专家将处理特定的数据样本。由于 MoE 层往往规模极大，因此通常需要对多个维度进行分片。

扩散模型。扩散模型采用一系列去噪自编码器进行图像合成。稳定扩散[36] 将 UNet 与注意力机制相结合，以支持根据文本等输入生成图像。该模型架构包含跨层连接，使流水线设计更加复杂。

DNABERT。DNABERT [16] 是 BERT[6] 模型的一个变体，针对启动子、剪接位点和转录因子结合位点等基因组相关任务进行了调优。与用于 NLP 任务的 BERT 模型相比，DNABERT 具有更长的序列长度，以捕获更长的核苷酸上下文。这促使人们在训练该模型时沿词元维度进行并行化，而现有研究很少探索这种方法。

2.4 生产模型

我们列出了生产集群中使用的一些模型。这些模型应用广泛，其训练性能对许多服务至关重要。具体而言，对于自然语言处理（NLP）任务，我们使用 GPT [3],T5 [31],BERT [6], ALBERT [20],和 RoBERTa [26],，它们集成在内部自然语言处理工具包中。对于与计算机视觉（CV）相关的任务，我们使用 VGG19 [41] 和 ResNet [13] 进行大规模人脸检测。此外，我们使用 M6 [25]处理推荐中的多模态任务。

3 概述

Rhino 是一个面向工业生产而设计的独立分布式训练系统，包括执行规划器和分布式运行时。如图 1 所示，执行规划器首先将单卡计算图（以 HLO IR 表示）和集群配置作为输入，然后对模型进行优化和分发，在每个设备上生成单程序多数据策略（第 4 节提及）、流水线策略（第 5 节详述）以及静态调度（第 6.2 节介绍）计划。随后，每个设备通过远程过程调用从执行规划器进一步获取其本地执行计划，并利用 Rhino 的运行时

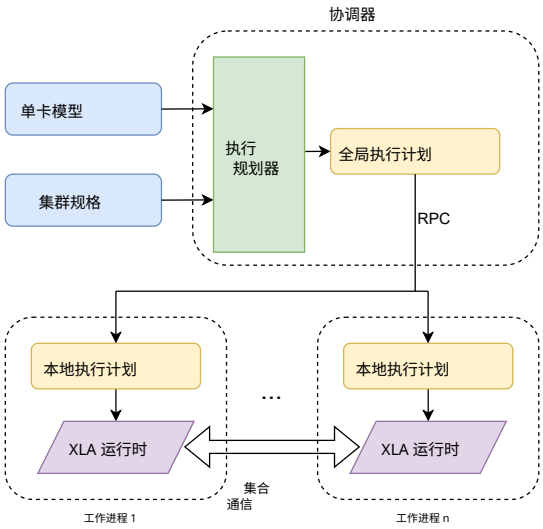


图 1：Rhino 概览

执行其在分布式程序中的部分工作。由于通过任务图机制（第 6.1 节）建立了提前静态调度，因此开销极低。

Rhino 执行规划器的工作流程如图 2 所示。我们枚举不同的设备网格配置，并通过单程序多数据策略搜索算法和流水线规划算法，为每种配置并行搜索策略。随后，我们从不同配置中选出最优策略，并执行静态调度。

Rhino面向工业生产，因此系统设计目标同时考虑编译时间和运行时性能。通常，对于包含大量指令的HLO模型，同时搜索单程序多数据和流水线策略是业界无法接受的。这要求合理设计自动并行化的探索方法。受深度模型在线反馈的启发，我们总结并提出了一种新颖的启发式搜索算法，在仅略微牺牲生成的并行化策略质量的情况下，显著缩小搜索空间并加速编译。对于单程序多数据策略，Rhino采用图粗化和三级子图合并算法，通过从原始图中识别关键节点和锥形结构，实现可接受的编译时间。随后，它将每个子图中的策略探索建模为整数线性规划问题，并使用动态规划联合优化所有子图。对于流水线策略，Rhino通过识别出的计算密集型操作对整个模型进行分段，同时不对拓扑顺序进行线性化。不同于Alpa [49], Rhino还基于一种结合界限收紧算法的启发式方法，将流水线探索建模为整数线性规划问题，从而显著缩短搜索时间。

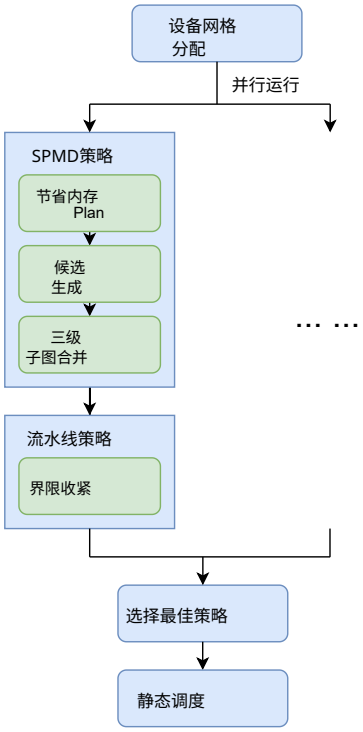


图 2：Rhino 执行规划器工作流程。

最后，我们开发了一个中央控制器，在单程序多数据策略、流水线策略或混合策略之间进行决策。所有启发式方法均采用数据驱动的设计，并包含加速编译时间所需的必要组件；与基于规则的方法相比，这有助于处理边界情况。

本文将在第4 节介绍我们的SPMD策略搜索算法，并在第5节介绍流水线策略。随后，我们将在第6节详述Rhino的运行时实现。在第7节，我们评估Rhino的性能；在第8节讨论相关工作；最后在第9节总结全文。

4 SPMD策略

为支持大型深度神经网络模型——其中单个张量可能超过一块GPU的内存限制——算子内划分是必要的。Rhino采用SPMD 并行，这使数据并行和模型并行成为可能。使用SPMD 并行时的主要挑战在于决定每个张量需要划分的维度，从而最小化总体通信时间。

SPMD分片可以在设备的正交子群上重复执行。例如，批量大小维度可以按机器进行分区，而隐藏维度则可以按机器内的 GPU 进行分区。在 Rhino 中，设备按设备网格组织，并在设备网格的每个维度上递归执行 SPMD 分片

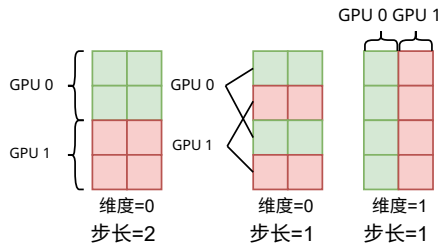


图 3：一个 4x2 张量的不同分片方法。

。我们会枚举不同的设备网格配置，并行运行相应的策略搜索算法，然后选出最佳策略。

4.1 张量分片规范

我们介绍 Rhino 中针对张量的各种分片方法。张量是一种多维数组。例如，图像分类模型的输入通常是一个三维张量，其第一维和第二维分别表示图像的宽度和高度，第三维表示颜色通道。

在分布式训练中，张量可以通过多种方式进行分片，称为分片规范。图3 展示了一个 4x2 张量的不同分片规范。在 Rhino 中，分片规范用一个元组(维度,步长)表示，其中维度定义要分区的张量维度，而步长表示分片时该维度中连续单元的数量。现有系统通常默认始终使用最大步长，即对应维度的大小除以分片数。不过，我们发现，对于包含重塑操作的某些模型，允许使用其他步长值可能会产生直观的分片策略。举例来说，重塑是对张量的元描述，不会改变其在内存中的布局，因此无需插入重新分片通信来连接与输入张量分区策略不兼容的操作。在我们的方法中，我们直接将重塑操作的步长值设为与其输入或输出相同的值。尽管重塑操作的步长值可能不兼容，但由于它不会影响数据布局，因此不会对操作产生副作用。这样，我们就能避免引入冗余的重新分片通信。

4.2 问题定义

我们首先分析一个简化的模型分片问题，它是 Rhino 设计的基础。

假设我们有一个包含n 个算子的模型，第i个算子有s_i 种不同的分片方法。目标是找到一种分片策略X，其中$X_{ij} = 1$ 表示为第i个算子选择第j种分片方法。

我们有两种成本。第一种是计算成本 comp^{ij}，即节点i 使用其第j种分片方法时的计算时间。第二种是通信成本 $\text{comm}_{i_1 i_2 j_1 j_2}$，指算子$i_1$ 和i_2之间存在连接，且它们分别使用j_1 和j_2 分片方法时的通信成本。

随后可以按如下方式计算总成本。

$$C = \sum_{i=1}^{n} \sum_{j=1}^{s_i} \text{comp}_{ij} X_{ij} + \sum_{(i_1,i_2)\in E} \sum_{j_1=1}^{s_{i_1}} \sum_{j_2=1}^{s_{i_2}} \text{comm}_{i_1 i_2 j_1 j_2} X_{i_1 j_1} X_{i_2 j_2}$$

然而，其中包含二次项 $X_{i_1,\ j_1} X_{i_2,\ j_2}$，无法直接使用整数线性规划求解器求解。因此，我们引入辅助整数变量 $A_{i_1 i_2 j_1 j_2} \geq X_{i_1 j_1} + X_{i_2 j_2} - 1$，并将优化问题改写如下（$[z]$表示集合 $\{0, 1, \ldots, z-1\}$）。

$$\min \sum_{i=1}^{n} \sum_{j=1}^{s_i} \text{comp}_{ij} X_{ij} + \sum_{(i_1,i_2)\in E} \sum_{j_1=1}^{s_{i_1}} \sum_{j_2=1}^{s_{i_2}} \text{comm}_{i_1 i_2 j_1 j_2} A_{i_1 i_2 j_1 j_2}$$

$$\begin{aligned}
\text{s.t.:} \quad & \sum_{j=0}^{s_i} X_{ij} = 1, && \forall i \in [n] \\
& A_{i_1 i_2 j_1 j_2} \geq X_{i_1 j_1} + X_{i_2 j_2} - 1, && \forall (i_1, i_2) \in E, j_1 \in [s_{i_1}], j_2 \in [s_{i_2}] \\
& X_{ij} \in \{0, 1\}, && \forall i \in [n], j \in [s_i] \\
& A_{i_1 i_2 j_1 j_2} \in \{0, 1\}, && \forall (i_1, i_2) \in E, j_1 \in [s_{i_1}], j_2 \in [s_{i_2}]
\end{aligned}$$

4.3 使用图粗化寻找单程序多数据策略

4.3.1 寻找关键节点

现代深度神经网络可能包含数千个操作。在这种规模下，求解第 4.2 节中的整数线性规划问题的最优解通常并不现实。为缩小搜索算法中的问题规模，必须采用图粗化技术。Rhino 开发了有效的图粗化启发式方法，并借鉴数据驱动反馈来辅助在线模型训练。观察多个在线模型后，我们得出了两个重要发现：首先，尽管包含许多操作，但少数计算密集型节点（例如矩阵乘法和卷积）占据了大部分计算时间。其次，大多数模型的计算图通常包含多个并发计算的部分，并由少数节点构成模型主干。通过识别主干上的这些计算密集型节点，并将它们用作分段边界，可以在不妨碍关键解的情况下，将整个图划分为若干子图，同时保持一定的粗化程度。主干上的这类计算密集型节点被标记为关键节点。

我们将模型的骨干路径定义为关键路径上的所有操作。为实现这一点，我们需要在逻辑上计算每个操作的最早和最晚计算时刻。我们不计算实时性，而是假设每个操作的执行时间

均为 1，且张量传输时间为 0。然后，可以根据拓扑顺序计算每个操作的最早时刻。相反，每个操作的最晚时刻则根据拓扑的逆序确定。具有相同最早时刻和最晚时刻的节点即构成骨干路径。最后，我们选择骨干路径上所有计算密集型节点作为关键节点，并采用这些节点对图进行粗化。

该方法基于大量在线反馈数据开发，并已在实践中证明有效。得益于数据驱动反馈，这一缩略策略未来仍有改进空间。

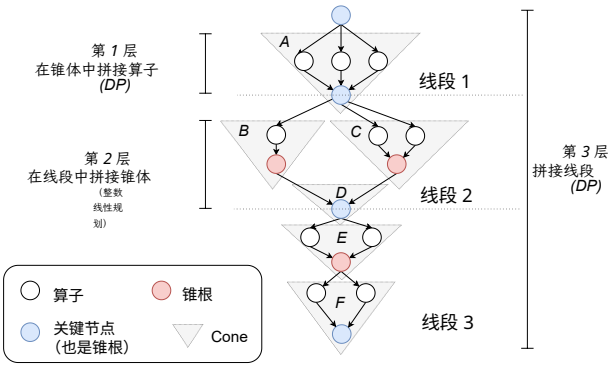


图 4：图上的分层分割及其对应的合并方法。

4.3.2 三级子图合并

我们提出了一种通过反复合并子图来推导单程序多数据策略的新方法。其核心思想是分而治之：如果能将计算图粗化为一个单节点，就可以通过枚举其分片方法，轻松确定最优策略。这种方法的挑战在于设计一种子图合并流程，既能系统地缩小图的规模，又能保留策略空间中的大部分内容。基于在生产集群中优化常见模型时的观察和统计数据，我们设计了一种三级子图合并算法，以高效且近乎最优地搜索单程序多数据策略。我们通过识别深度神经网络模型中的特殊结构，即锥形结构和关键节点，将整个图划分为多个部分，并使用不同算法合并这些子图。由于我们识别的是图结构而非特定算子类型，因此该方法可推广到新的深度神经网络模型。另一方面，我们的方法利用了深度神经网络模型中常见的特殊结构，优于那些忽视深度神经网络训练特点的通用启发式方法。

我们首先介绍在 Rhino 中将子图合并为单个节点的步骤。子图表示为一组节点。割张量 是连接子图内部节点与子图外部节点的张量。将子图合并为单个节点，需要移除子图内部的算子和张量，在计算图中插入一个新的抽象算子，并将所有割张量连接到该抽象算子。此外，粗化抽象算子的候选策略在子图合并后确定，这取决于合并算法。

现在，我们介绍三级子图合并方法，该方法经过系统设计，用于搜索单程序多数据策略。从宏观上看，Rhino 将模型图按层次划分为三个级别。在每个层次中，Rhino 通过动态规划或整数线性规划来确定单程序多数据策略，如图4所示。对于底层（记为第 1 级），我们应识别锥形结构和锥根。锥根是具有多个输入的算子。锥形结构是一种特殊子图，仅由一个锥根和其他节点组成。我们观察到，大多数

我们生产集群中的深度神经网络模型都包含锥形结构。为识别锥形结构，Rhino 会在整个图中查找所有锥根，然后不断扩展每个锥根的输入，直到遇到另一个锥根。锥形结构具有一个有利特性：在锥内，除锥根外的所有节点都只有一个输入张量；否则，该节点本身就是锥根。这使得我们能够利用动态规划，为每个算子拼接策略。给定锥根已枚举的策略后，Rhino 会通过最小化重新分片引入的通信开销，优化锥内所有其他节点的最优策略。因此，锥的策略由给定的锥根策略决定。在这一层次中，对于每种枚举情况，我们都会保留锥的所有可能策略。在中间层次中，由于图由锥形结构而非细粒度节点组成，问题规模已大幅缩小。Rhino 将锥合并问题建模为整数线性规划问题，以处理不同策略的锥。

然而，锥的数量仍可能非常庞大，导致整数线性规划求解器无法在可接受的时间内完成求解。这是大型模型中经常遇到的挑战。Rhino 引入了顶层层级结构和另一种启发式方法，以进一步降低问题复杂度。它以关键节点为枢轴，将整个图划分为若干近似线性片段，如图4 所示。由此，模型图被简化为带有可能策略的简单线性结构，并可通过动态规划轻松拼接。在每个片段内，Rhino 使用整数线性规划求解器确定其策略，并利用动态规划求解锥策略。引入顶层层级分段后，策略质量不一定能够得到保证。然而，如果策略质量不会受到太大影响，那么缩短探索时间是合理的。Rhino 提供了两个优化级别，区别在于是否引入顶层层级分段；这有助于降低锥合并的问题复杂度。针对不同场景，它在策略质量和搜索时间之间提供了灵活的权衡。

4.3.3 候选生成

采用我们的步长分区方法（第 4.1 节）后，即使使用我们的三级子图合并算法，策略空间仍然过大，难以高效搜索。例如，要将一个 256x256 的张量均匀地放置到 4 台设备上，共有$2*$个因数$(256/4)+2=16$种不同的方法，其中前两个 2 表示沿两个维度进行分片；因数$(256/4)$表示在某个维度上分片时可能的步长数量，这等于每台设备上该维度大小的因数数量；最后一个 2 表示非分片方法，即复制和部分分片。采用这两种方法时，张量形状保持完整，但每台设备上的结果是部分结果。

我们观察到，大多数分片规范所生成的策略具有相同的性能。例如，在整个模型中使用不同的步长，会得到计算量和通信量相同的策略。因此，我们提出默认让所有张量使用最大步长，仅在能够减少通信时才使用其他步长。为此，我们基于这一理念设计了一种候选生成算法。

从计算密集型算子（例如 矩阵乘法）开始，我们枚举其分片方法，其中所有涉及的张量均使用最大步长。然后，我们使用无通信合并算法（第 4.3.2节）递归地将其与相邻算子合并，直到无法在不引入通信的情况下继续合并相邻算子。在此过程中，我们将每个算子所使用的所有分片方法记录为候选项。通过将搜索限制在这些候选项内，我们可以避免探索不会带来更优策略的分片规范。

4.4 内存约束

在训练参数量达数十亿的大型模型时，GPU 内存限制是一个主要问题，也是采用 SPMD 并行的动因之一。通过对参数张量进行分片，并将互不重叠的切片存储在不同设备上，可以减少 GPU 上的内存使用量。然而，分片通常会引入通信，从而延长训练时间。现有的节省内存技术（如 ZeRO） [33], 会应用于整个计算图。因此，它们往往会对超出必要范围的张量进行分片，导致不必要的通信开销。

Rhino 提出了一种算法，可自动寻找在确保内存使用量不超过设备限制的同时，最大限度缩短训练时间的 SPMD 策略。该算法将内存使用量视为约束，而不会对其进行过度优化。因此，它可能产生难以手动设计的复杂策略。例如，当内存限制不足以复制所有参数时，它可能只对部分大型张量应用 ZeRO，而复制其他张量。

我们的算法基于这样一种观察：少数张量占用了大部分内存。因此，对大型参数进行分片可以节省内存，同时不会对生成的策略产生太大影响。具体而言，我们首先按大小降序排列模型中的参数，并标记前k 个参数。然后，我们移除会复制这些张量的分片候选项，强制在策略搜索期间对它们进行分区。我们选择能够提供可行解的最小k 。

5 流水线策略

5.1 问题表述

采用单程序多数据并行时，每个设备都会在分片张量上运行相同的程序。在大型集群上训练时，每个设备上的分片张量可能会变得过小，无法充分利用计算单元，导致 GPU 利用率不足。流水线并行可以与单程序多数据并行互补，用于训练极深的模型。

采用流水线并行时，模型会被划分为多个阶段。随后，每个阶段会分配给一组设备，并进一步通过单程序多数据并行在这些设备上进行分区。Rhino 无需按线性顺序对所有指令进行排序，而是利用每个节点的拓扑关系直接对图进行分段，从而保证策略质量。在 Rhino 中，我们考虑沿设备网格的一个维度进行流水线并行，以统一流水线并行与单程序多数据并行的策略空间。

假设我们在大小为 d 的设备网格维度上采用流水线并行。第一步是将模型划分为 d 个阶段。我们通过求解以下优化问题来确定阶段分配方案，使通信量最小化，同时保持各阶段负载均衡。

流水线并行可以表述为整数线性规划问题。我们使用整数变量 B_i 表示算子i 被分配到的阶段。对于计算图中的每条边（i, j），都有 $B^j - B_i \geq 0$，因为张量的消费者应被放置在不早于其生产者的阶段。当一对生产者和消费者被放置在不同阶段时，即 $B_j \neq B_i$，就需要通过通信在设备之间传输张量（i, j）。对于每条边（i,j）$\in$ E，我们引入一个 0-1 变量 F^{ij}，用于表示指令i 和j 是否被放置在不同阶段。我们使用大 M 法 [11]，通过添加以下约束来定义 F^{ij}：

$$F_{ij} \leq B_j - B_i \leq F_{ij} \cdot M$$

其中 M $\geq$ d $-$1 是一个足够大的常数。当 i 和 j 被放置在同一阶段时，即 $B_i = B_j$，F_{ij}只能取 0，因为 $F_{ij} \leq B_j - B_i$。当 i 和 j 被放置在不同阶段时，F^{ij}必须为 1，因为 $B_j - B_i > 0$且 $B_j - B_i \leq F^{ij} \cdot M$。M $\geq$ d $-$1 确保当 $F_{ij} = 1$ 时，$B_j - B_i \leq F^{ij} \cdot M$ 始终成立，因为只有 d 个阶段，因此 $B_j - B_i \leq$ d $-$ 1。

定义 F^{ij} 后，寻找流水线并行度的目标可以轻松地建模为最小化总通信量，如下所示：

$$\sum_{(i,j)\in E} F_{ij} \cdot \text{tensor_size}(i,j)$$

5.2 界限收紧

第5.1 节所示的公式可能会带来令人望而生畏的问题规模，因为其中包含大量不平衡的阶段划分。我们提出一种界限收紧技术，以限制每个阶段的最大不平衡容差，从而高效地求解该优化问题。对于计算图中的给定指令 i，我们定义两个指令集 A_i 和 D_i，它们分别是 i 的祖先和后代。i 的祖先是指在计算图中存在一条路径通向i 的指令。类似地，i 的后代是指其祖先包含i的指令。图 5 展示了集合A_i 和D_i。由于数据依赖，A_i 中的指令必须在执行i 之前完成。因此，累积执行时间 $\sum_{j\in A_i} comp_j$ 是i最早的开始时间。通用指令集记为M。为满足平衡约束，我们有

$$B_i \geq \lfloor \frac{\sum_{j\in A_i} comp_j}{time_per_stage} - \varepsilon \rfloor$$

其中，ε 是一个超参数，用于指定各阶段之间计算不平衡的容差水平。$时间_每_阶段 = \sum_{i\in M} comp_i/d$ 是在完全平衡的假设下每个阶段的计算时间。反过来，我们也可以根据D_i定义i 最晚的开始时间：

$$B_i \leq \lceil \frac{comp_i + \sum_{j\in D_i} comp_j}{time_per_stage} + \varepsilon \rceil$$

为说明这一点，这两个约束将指令 i 限制在容差范围内，安排于完美平衡阶段的前一阶段或后一阶段。将这些约束纳入整数线性规划表述后，即可高效求解优化问题。在 Rhino 中，用户可以设定容差级别，以控制探索时间。

6 实现

6.1 任务图

Alpa 等现有系统 [49] 采用 MPMD 风格运行时来编排流水线执行，并构建用于管理内存、同步以及利用驱动进程调度指令的指令。在 Rhino 中，我们抽象出任务图来安排策略执行。任务图是由以下基本元素构成的有向无环图（DAG）。

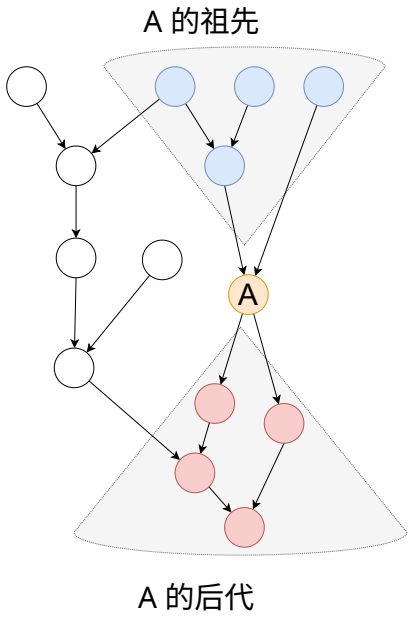


图 5：指令 A 的祖先和后代以不同颜色表示。Rhino 直接依据拓扑关系对图进行分段，而非按线性顺序对节点排序。

- **任务节点** 任务节点用于表示在设备间执行的计算或 Send/Recv 点对点通信。需要流水线时，每个任务节点都关联一个阶段 HLO 模块，可将其视为程序，而任务节点则表示该程序的实例。对于需要多个 SPMD 风格任务共享同一 HLO 模块的应用，可通过使用不同的输入或参数来区分这些任务。

- **源节点和汇节点**。它们分别表示任务图的开始节点和结束节点。
- **边**。它表示任务节点之间一系列张量的传输。

任务图为组织和执行任务节点之间的计算或通信提供了直观的方式。通常，这类计算由 DAG 描述，其中包含多个任务节点。多个任务节点可以共享同一个 HLO 模块，并以不同的输入或参数作为输入，从而实现单程序多数据并行。通过按顺序连接多个任务节点，也可以实现流水线并行。借助这些任务图，还可以自动规划节省内存调度和梯度检查点。采用不同策略的任务图如图6所示。

6.2 静态调度

静态调度是一种并行计算形式，涉及将预定义的任务分配给不同的处理单元。与动态调度相比，后者涉及

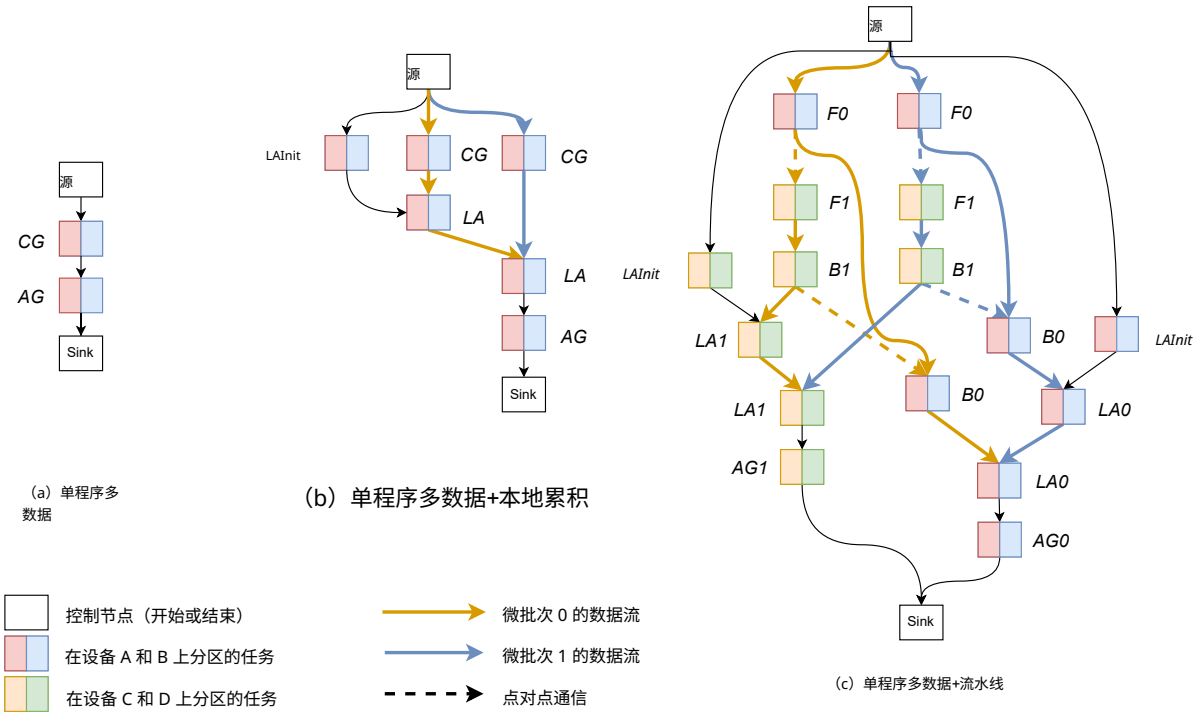


图6：采用不同策略的任务图。不同颜色表示不同的设备。具有多种颜色的节点表示其跨多个设备进行分片。蓝线和黄线分别表示不同的微批次。虚线表示跨设备的点对点通信，需要插入发送/接收对；实线则不需要。对于纯单程序多数据策略（a），任务图仅包含计算梯度（CG）和应用梯度（AG）。对于结合多个微批次的单程序多数据策略（b），应创建本地累积（LA）。对于单程序多数据结合流水线策略的情况，还应为相应的反向阶段创建本地累积。

在任务运行时动态分配任务，静态调度的性能更好，因为任务顺序和内容都已预先确定，无需依赖主控节点来决定接下来要做什么。目前，1F1B [8,28] 调度顺序已在 Rhino 中实现为默认方案，事实证明，这对于提升吞吐量并节省内存是必要的。具体而言，基于任务图的静态调度更易于实现，也有望探索出更优方案。这是因为任务图粒度较粗，相较于 HLO图，其调度搜索空间小得多，从而缩短了探索所需的时间。

6.3 分布式初始化与通信器管理

训练大型模型时，变量初始化非常耗时。为简化这一过程，Rhino 直接在分配的设备上实施分布式初始化，而不是先在主节点上进行初始化，再将相关分片复制到目标位置。初始化期间的数值正确性取决于根据张量的分片信息跳转到正确的种子状态。

Rhino 的运行时能够同时运行多种并行策略这使通信器的管理变得非常复杂。为简化这一过程，我们会在训练开始前，为当前训练模型创建并协商所有必需的通信器。我们收集通信指令以及 Send 和 Recv 任务节点，并尝试创建合适的通信器。为避免重复，我们维护一个全局通信器缓存。

7 评估

7.1 实验设置

测试平台。我们在两个采用典型商用硬件的集群上进行实验。

- **M8 平台：**（每台机器）8 * V100-SMX2-32GB GPU，配备 NVLink2；2 * Xeon（Skylake）CPU，48 核，2.5GHz；768GB DDR4-2666；1 * 100G RoCE 互连。
- **平台 S1：**（每台机器）1 * V100S-PCIE-32GB GPU；2 * Xeon（Cascade Lake），52 核，2.5GHz；512GB DDR4-2666；1 * 100G RoCE 互连。

所有机器均安装了 Linux内核 4.19.91 版、Nvidia驱动 470.82 和 NCCL库 2.8。对于

表 1：GPT 模型弱扩展性测试配置。完全并行测试的全局批次大小设为 64，而仅 SPMD 测试的批量大小设为 4。仅 SPMD 的 1–8 卡强扩展性测试使用“GPT-3 中型”配置。

配置名称	N_{params}	N_{layers}	D_{hidden}	D_{ffn}	N_{heads}	D_{head}	N_{GPUs}
GPT-Medium	350M	24	1024	4096	16	64	1
GPT-Large	760M	24	1536	6144	16	96	2
GPT-XL	1.3B	24	2048	8192	32	64	4
GPT-2.7B	2.7B	32	2560	10240	32	80	8
GPT-6.7B	6.7B	32	4096	16384	32	128	16

表 2：GShard MoE 混合专家模型弱扩展测试配置。全并行测试的全局批次大小设为 256，而仅 SPMD 测试的批量大小设为 8。

配置名称	N_{params}	N_{layers}	D_{hidden}	D_{ffn} 函数	N_{heads}	D_{head}	$N_{experts}$	N_{GPUs}
MoE-8E-380M	350M	8	768	6144	16	48	8	1
MoE-8E-760M	760M	16	768	6144	16	48	8	2
MoE-16E-1.3B	1.3B	16	768	6144	16	48	16	4
MoE-16E-2.4B	2.4B	16	1024	8192	16	64	16	8
MoE-32E-10B	10B	16	1536	12288	16	96	32	16

不同的基线系统，由于软件需求不同，使用了不同版本的 CUDA 工具包（Alpa/DeepSpeed：11.4，Rhino：10.1）。

基准测试模型。由于基线系统对其他模型的开源实现有限，我们主要在与基线进行比较时，针对两种自然语言处理模型开展实验：GPT [3] 和 GShard MoE [40],。这些模型所用的配置列于表 1和表 2。我们选择的配置与相关研究所用配置相同。我们还使用 VGG19 [41],DNABERT [16], UNet [37],和 Wide-ResNet [48]对 Rhino 进行了评估。

基准系统。我们将 Rhino 与两个最先进的系统 Alpa [49] 和 Megatron/DeepSpeed [29, 34]进行了比较。Alpa 是一个自动化分布式训练系统，支持单程序多数据并行和流水线并行。Megatron/DeepSpeed 指的是在 DeepSpeed 上运行 Megatron 专家设计的并行化策略和经过优化的 Transformer 实现。

所有实验均采用完整的 FP32（单精度）计算。

7.2 可扩展性

我们在 M8 平台上使用最多 16 个 GPU，对 GPT 和混合专家模型评估 Rhino 与基准系统的可扩展性。结果如图 7所示。

该 全自动 并行性是 Rhino 的核心设计标准，使其能够探索数据、张量（或分片）和流水线并行混合的最高效策略。通过在 1 至 16 个 GPU（M8 上的 2 个节点）之间进行扩展测试，验证了系统的能力；测试采用了针对 GPT 和 GShard MoE 的全并行策略规划，结果如图 7 所示。相同的工作负载也在同一平台上使用 Alpa 和 DeepSpeed 运行，结果如图所示。对于 Alpa，我们使用“PipeShardParallel”，并尝试在其提供的示例中寻找基准测试设置。对于 DeepSpeed，我们测试了若干可行的并行方案设置，并记录其中的最佳方案。

从结果可以看出，Rhino 驱动的 GPT 和混合专家模型训练，无论是在单个节点内（1 至 8 个 GPU）还是跨两个节点，都能良好扩展。我们按照 [29] 中的方法，测量了每项测试实际达到的浮点运算次数。考虑到 V100-SMX GPU 的理想单精度峰值性能为 15.6 TFLOPs，GPT 模型训练的 MFU [19] 为 54.0% 至 75.9%，混合专家模型训练最高可达 55.1%。与另一种全自动并行化系统 Alpa，以及最先进的手动并行化语言模型训练框架 Megatron（在 DeepSpeed 中采用）驱动的测试相比，许多情况下 Rhino 的性能与 Alpa 和 DeepSpeed/Megatron 相当，有时甚至略胜一筹。

7.3 消融研究

实际生产平台的统计数据表明，相当一部分并行训练任务使用的 GPU 少于 8 张。尽管我们的系统架构是为 SPMD+PP 并行性设计的，但也可以考察它在仅 SPMD 场景下对这些常见模型和工作负载的表现。

我们按照表 1和表2中的配置，对 GPT 和混合专家模型进行了仅 SPMD 弱扩展测试。此外，我们还对 GPT-Medium 配置进行了强扩展测试，以便更好地了解 SPMD 策略自动规划的结果。我们也使用 Alpa 和 Megatron/DeepSpeed 系统进行了相同的测试。对于 Alpa，在仅 SPMD 测试场景中使用了默认的“Shard-Parallel”方法。对于 Megatron/DeepSpeed 场景，我们仔细设置了手动调优的动态规划和模型并行参数，以获得最佳结果。大多数测试均在 M8 平台和平台 S1 上进行。考虑到实际生产环境中存在大量调度碎片，平台 S1 每个节点仅包含一张 GPU 卡，能够很好地反映这种情况。由于在低带宽互连下，仅采用流水线并行可能是合理的替代方案，我们还对 GPT 模型进行了仅 PP 弱扩展测试。

所有仅采用单程序多数据和仅采用流水线并行的测试结果如图 8 所示。结果证明，在单节点仅采用单程序多数据并行时，Rhino 可帮助语言模型获得较高的 MFU：在 M8平台上，GPT 的 MFU 最高可达 68.1%，混合专家模型则可达 62.0%。我们还将在下一小节通过更多案例展示良好的可扩展性。所有这些结果都表明，Rhino 能够轻松支持来自不同应用领域的多种模型。

从 GPT-Medium 模型配置的强扩展性测试结果可以看出，随着运行设备数量的变化，Rhino 的单程序多数据搜索结果也有所不同：

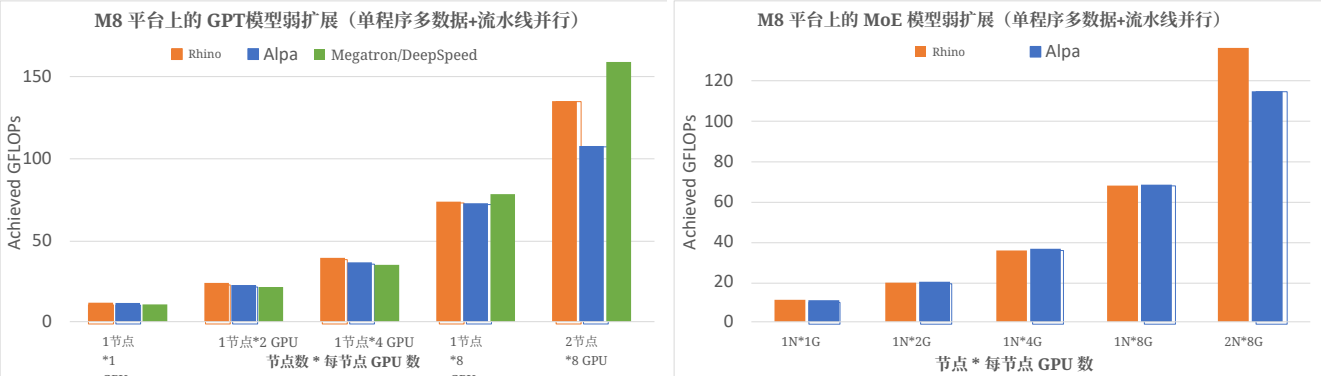


图 7： 完全并行时，GPT 和混合专家模型在 M8 平台上的性能（按参数规模划分）.

使用 2 个 GPU 时，策略与常规动态规划相同。使用 4 个 GPU 时，“解码器第 0 层”中的一个变量在设备之间进行分片，并且报告的数据传输量略有减少。设备数量增至 8 个时，所有“注意力查询/键/值”变量及其优化器状态变量都像 DeepSpeed Zero-2 一样被拆分 $Pos{+}g$” [34] 架构，而模型中的其他部分则仍采用动态规划。搜索报告显示，与动态规划相比，数据传输量减少了 20%。在 M8 平台和 S1 上，这些策略下的运行性能验证了搜索结果。相比之下，Alpa 在 8 个 GPU 上给出的方案更接近动态规划。

7.4 不同模型上的泛化能力

除了 GPT 和 GShard MoE 模型外，我们还纳入了另外四种模型来验证其普适性：1）VGG19 [41],，一种常用于图像识别的卷积神经网络。我们将模型的最后一层替换为百万级类别分类器。2）DNABert [16],，一种采用 Transformer 和卷积神经网络来解读 DNA序列的科学模型。3）Wide- ResNet [48],，一种通道规模更大的 ResNet 变体。我们随 GPU 数量增加模型规模，以研究其可扩展性。4）Unet，流行的扩散网络中的关键组件。对于 Wide-ResNet，我们随 GPU 数量增加模型规模，以研究其浮点运算次数。对于其他模型，我们随 GPU 数量增加批量大小，以研究其吞吐量。模型的多样性有助于我们更全面地考察 Rhino 的普适性。实验配置列于表 3和表 4。

所有评估结果如图 9所示。一方面，两组扩展数据实验都实现了接近线性的可扩展性。对于 VGG19，分类器会消耗大量设备内存，从而阻碍在单个设备上进行训练。但借助 Rhino 探索出的合适并行策略，它能够实现扩展。对于较小的模型，如 DNABert 和 UNet，Rhino 更倾向于采用简单的数据并行。另一方面，Wide-ResNet 在参数规模上实现了扩展，同时也达到了近线性的吞吐量。

表3：VGG19 和 DNABERT 基准测试配置。

模型名称	N_{params}	批量大小/设备	N_{GPUs}
VGG19	4.2B	128	从 1 到 8
DNABERT	162M	32	从 1 到 8
UNet	33M	16	从 1 到 8

表4：Wide-ResNet 基准测试配置。我们将每次实验的全局批量大小保持为 4。

配置名称	N_{params}	$N_{channels}$	$N^{width_{factor}}$	N^{layers}	N_{GPUs}
WR-1.3B	1.3B	320	2	50	1
WR-2.6B	2.6B	448	2	50	2
WR-6B	6B	640	2	50	4
WR-12B	12B	320	12	101	8

7.5 SPMD策略搜索性能

为了说明单遍单程序多数据策略规划时间如何受到训练模型复杂度和端到端性能的影响，我们使用另一组定制配置进行实验：设置 GPT 模型骨干（将隐藏层大小设为 512，然后针对不同情况调整层数），随后在 1～8 个 GPU 上运行仅采用单程序多数据的并行案例。我们记录了策略规划时间（在 2个GPU 上）以及采用数据样本弱扩展时的训练步骤迭代时间。在此案例中，我们分别测试了整个计算图级搜索（OPT_LEVEL_3，O3）和子图级搜索（OPT_LEVEL_2，O2）。结果如图 10所示。

结果表明，当模型层数超过36层、图中包含超过3万条HLO指令时，O2级别方法的搜索时间优于全图搜索（O3）。但无论从引擎报告还是M8平台上的端到端性能来看，O2级别方法给出的单程序多数据策略与O3相比仍存在10%的差距。除了数据传输量的差异外，性能下降还可能

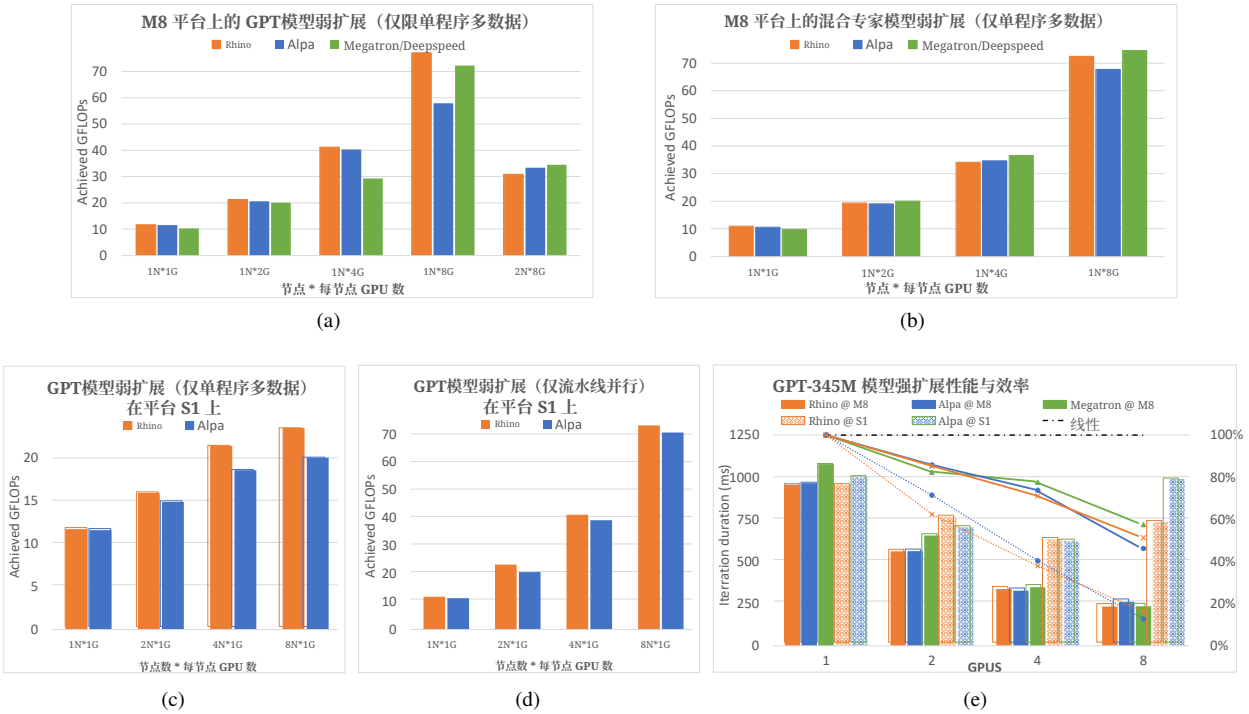


图 8：仅采用单程序多数据或流水线并行时 GPT 和混合专家模型的扩展性能。(a)-(d) 为表中提到的参数弱扩展案例1，(e) 为 GPT-Medium 配置下的强扩展。

表 5：GPT-72层和 MoE-32E-10B 中关键节点和子图的统计信息。

模型名称	指令	关键节点	比例	子图数量
GPT-72-Layers	76k	378	0.5%	377
MoE-32E-10B	18k	90	0.5%	89

归因于O2搜索增加了对AlltoAll通信的使用。然而，这种集合通信性能可能无法高效利用可用的通信带宽。

通过将原始图划分为多个粒度更粗的子图，可以节省搜索时间。划分过程取决于关键节点的识别。我们收集在线统计数据，并与原始操作数量进行比较，结果表明关键节点的占比很小。这直接有助于缩短搜索时间，对大型模型尤为显著。表 5 展示了两个具有代表性的大型模型的统计数据。结果表明，在 GPT-72-Layers 和 MoE-32E-10B 中，关键节点的占比均约为 0.5%。因此，原始大图的规模从 76k 和 18k 分别缩减至 377 和 89。这项优化大幅缩短了编译时间，而运行时的性能损耗仅为 10%，这对于实际生产应用具有重要意义。

8 相关工作

数据并行系统。PyTorchDDP [23] 是基于 PyTorch 的数据并行训练系统，使用全归约同步梯度。Distribution Strategy 提供参数服务器，通过异步梯度下降训练稀疏模型。Horovod [38] 将数据并行优化器设计为第三方插件。它利用贝叶斯优化器调整张量通信组的大小，从而提供自动调优方法。ZeRO [33]通过分区存储状态变量来提高内存使用效率，从而便于训练大型模型。

模型并行系统。Mesh-TensorFlow [39] 提供单程序多数据风格的模型并行接口来构建模型。 [47] 依赖用户对 HLO 的注释来获取单程序多数据计划。GPipe [15] 将深度模型划分为多个阶段，并以流水线方式将其并行化。PipeDream [28]通过采用异步训练改进流水线并行，并将其与数据并行相结合。DAPPLE [8] 在同步训练中采用 1F1B 调度。TeraPipe [24] 发现了 Transformer 语言模型在令牌级维度上的流水线并行性。

手动设计的并行化策略。Megatron[29] 支持通过专家设计的并行化策略大规模训练 Transformer模型，这些策略结合了数据并行和模型并行。DeepSpeed [34] 进行了优化

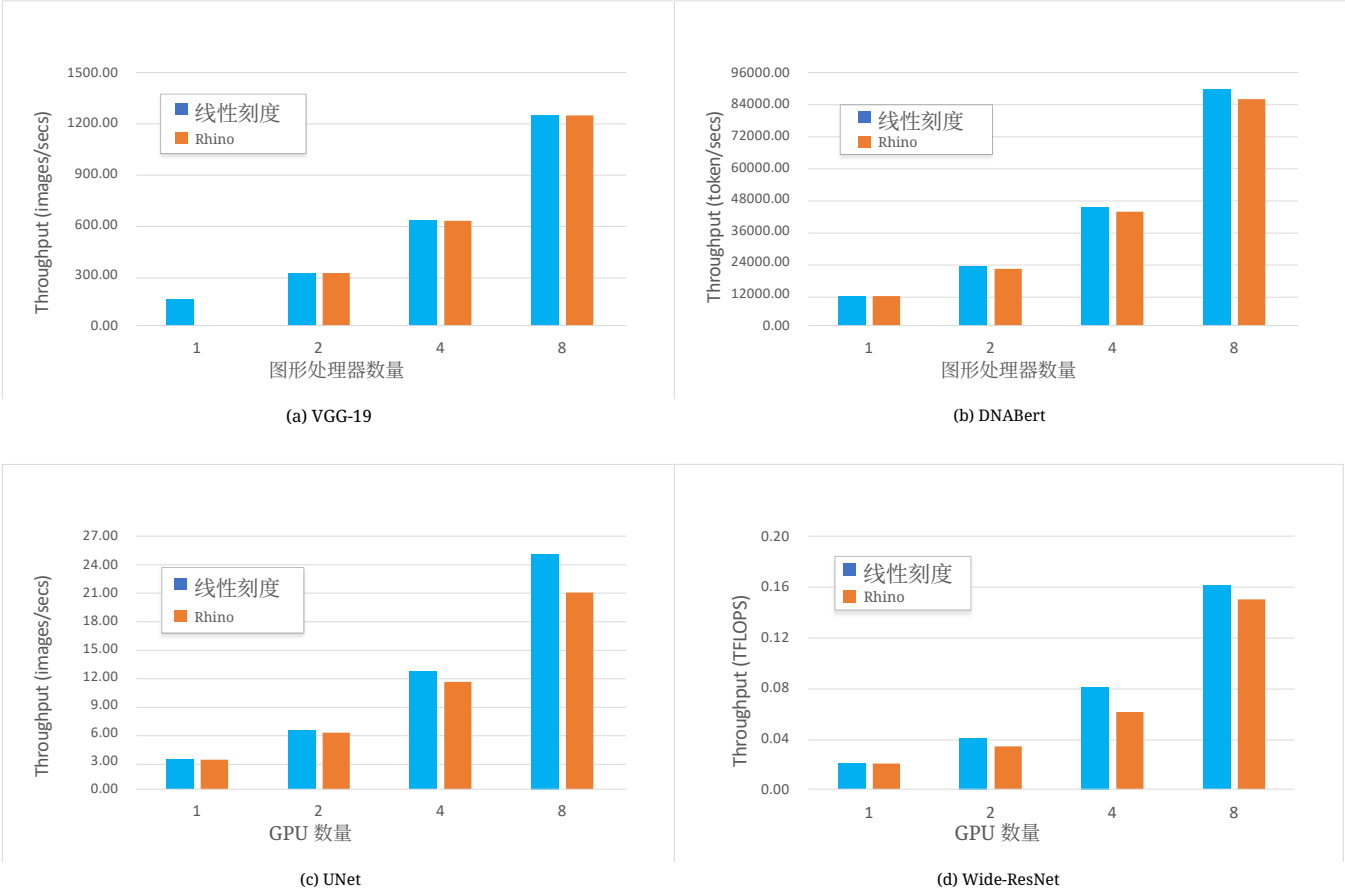


图 9：M8 单节点上其他通用模型的弱扩展评估。

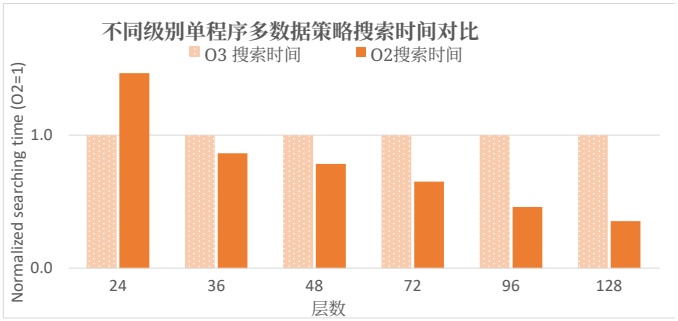
内核实现以及专家精心设计的并行化策略。DeepSpeed-MoE [32] 改进了混合专家模型的架构，并为训练和推理模型设计了并行化方案。FasterMoE [12] 引入了实现动态负载均衡并优化拥塞全互连通信的技术，以加速 MoE 模型训练。

自动并行化系统。Alpa [49] 以层次化方式制定模型并行化，将算子内并行性建模为整数线性规划问题，并设计动态规划方法来探索算子内和算子间的并行性。Unity [44] 引入并行计算图，并在图上使用随机图替换，以联合优化代数变换和并行化。FlexFlow [18] 定义了一个“SOAP”搜索空间，并通过随机搜索探索并行化。Whale[17] 遵循计算均衡原则对模型进行划分，以适配异构计算设备，并允许通过并行化原语指定并行化策略。Tofu [45] 通过递归搜索最小化通信时间，并利用区间分析自动发现并行化维度。PlaceTo [2] 利用强化学习为模型并行决定设备放置。

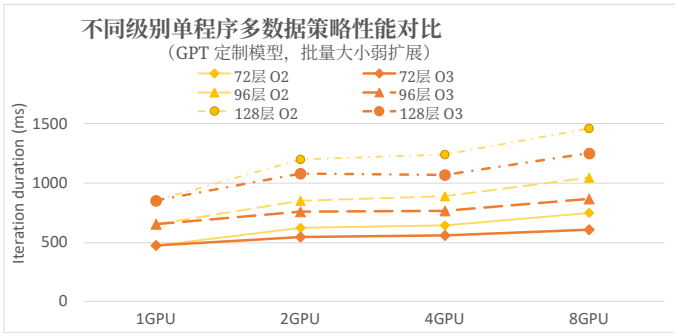
深度学习编译器。MLIR [21] 是一种分层且可扩展的编译器基础设施，它对基于静态单赋值的中间表示数据结构进行了标准化。Relay [35]提出了一种编译器框架，用于统一并泛化现有框架中的中间表示。TVM [4] 是一种编译器，通过提供图级和算子级优化，实现深度学习工作负载在多种硬件后端上的性能可移植性。

9 结论

我们介绍了 Rhino，这是一套通过自动并行化加速张量程序的工业系统。为实现高效且通用的单程序多数据策略搜索，我们设计了三级子图合并算法，并结合构建的整数线性规划和动态规划方法来寻找解决方案。对于流水线并行，非线性模型的流水线并行阶段划分被建模为整数线性规划问题，并通过界限收紧启发式方法加速。所有启发式方法均采用数据驱动的——



(a)



(b)

图 10：不同级别的单程序多数据策略搜索对比。“O3”表示完整图整数线性规划方法，“O2”表示子图整数线性规划加动态规划方法。

设计。它们在缩短探索搜索时间方面发挥了关键作用，从而推动自动并行化向通用化发展，并应用于工业人工智能平台。最后，我们实现了一个基于任务图的静态调度的分布式运行时引擎，其运行时开销几乎无法观测。

参考文献

[1] Martín Abadi、Paul Barham、Jianmin Chen、Zhifeng Chen、Andy Davis、Jeffrey Dean、Matthieu Devin、S an- jay Ghemawat、Geoffrey Irving、Michael Isard 等。TensorFlow：一种面向大规模机器学习的系统。发表于第 *12* 届 {*USENIX*} 操作系统设计与实现研讨会（{*OSDI*} *16*），2016年。

[2] Ravichandra Addanki、Shaileshh Bojja Venkatakrishnan、Shreyan Gupta、Hongzi Mao 和 Mohammad Alizadeh。PlaceTo：为分布式机器学习学习可泛化的设备放置算法。*arXiv* 预印本 *arXiv:1906.08879*，2019年。

[3] Tom B Brown、Benjamin Mann、Nick Ryder、Melanie Subbiah、Jared Kaplan、Prafulla Dhariwal、Arvind Nee- lakantan、PranavShyam、GirishSastry、AmandaAskell 等。语言模型是少样本学习者。*arXiv* 预印本 *arXiv:2005.14165*，2020年。

[4] Tianqi Chen、Thierry Moreau、Ziheng Jiang、Lianmin Zheng、Eddie Yan、Haichen Shen、Meghan Cowan、Leyuan Wang、Yuwei Hu、Luis Ceze 等。 {TVM}：用于深度学习的自动化 {端到端} 优化编译器。发表于第 *13* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 18*），第 578–594 页，2018年。

[5] Aakanksha Chowdhery、Sharan Narang、Jacob Devlin、Maarten Bosma、Gaurav Mishra、Adam Roberts、Paul Barham、Hyung Won Chung、Charles Sutton、Sebastian Gehrmann 等。PaLM：借助 Pathways 扩展语言建模规模。*arXiv* 预印本 *arXiv:2204.02311*，2022年。

[6] Jacob Devlin、Ming-Wei Chang、Kenton Lee 和 Kristina Toutanova。BERT：用于语言理解的深度双向 Transformer 预训练。*arXiv* 预印本 *arXiv:1810.04805*，2018年。

[7] Nan Du、Yanping Huang、Andrew M Dai、Simon Tong、Dmitry Lepikhin、Yuanzhong Xu、Maxim Krikun、Yanqi Zhou、Adams Wei Yu、Orhan Firat 等。GLaM：通过混合专家高效扩展语言模型规模。载于国际机器学习会议，第 5547–5569 页。PMLR，2022年。

[8] S. Fan、Y. Rong、C. Meng、Z. Cao 和 W. Lin。DAPPLE：一种用于训练大型模型的流水线式数据并行方法。发表于 *PPoPP '21*：第 *26* 届 *ACM SIGPLAN* 并行编程原理与实践研讨会，2021年。

[9] William Fedus、Barret Zoph 和 Noam Shazeer。Switch Transformer：通过简单高效的稀疏性扩展至万亿参数模型。*arXiv* 预印本 *arXiv:2101.03961*，2021年。

[10] Roy Frostig、Matthew James Johnson 和 Chris Leary。通过高级跟踪编译机器学习程序。机器学习系统，4(9)，2018年。

[11] Igor Griva、Stephen G Nash 和 Ariela Sofer。线性与非线性优化，第 108 卷。SIAM，2009年。

[12] Jiaao He、Jidong Zhai、Tiago Antunes、Haojie Wang、Fuwen Luo、Shangfeng Shi 和 Qin Li。FasterMoE：大型动态预训练模型训练的建模与优化。发表于第 *27* 届 *ACM SIGPLAN* 并行编程原理与实践研讨会论文集，第 120–134 页，2022年。

[13] 何恺明、张祥雨、任少卿和孙剑。用于图像识别的深度残差学习。载于*IEEE*计算机视觉与模式识别会议论文集，第770–778页，2016年。

[14] Geoffrey Hinton。《前向-前向算法：一些初步研究》。2022年。

[15] Yanping Huang、Youlong Cheng、Ankur Bapna、Orhan Firat、Dehao Chen、Mia Chen、HyoukJoong Lee、Jiquan Ngiam、Quoc V Le、Yonghui Wu 等。《Gpipe：利用流水线并行高效训练巨型神经网络》。神经信息处理系统进展，32：103–112，2019年。

[16] Yanrong Ji、Zhihan Zhou、Han Liu 和 Ramana V Davu-luri。DNABERT：用于基因组 DNA语言的预训练双向编码器Transformer模型。生物信息学，37(15):2112–2120，2021年。

[17] Xianyan Jia、Le Jiang、Ang Wang、Wencong Xiao、Ziji Shi、Jie Zhang、Xinyuan Li、Langshi Chen、Yong Li、Zhen Zheng 等。Whale：基于异构｛图形处理器｝的高效巨型模型训练。载于*2022 USENIX*年度技术会议（*USENIX ATC 22*），第673–688页，2022年。

[18] Zhihao Jia、Matei Zaharia 和 Alex Aiken。超越深度神经网络的数据并行和模型并行。*arXiv*预印本 *arXiv:1807.05358*，2018年。

[19] Vijay Anand Korthikanti、Jared Casper、Sangkug Lym、Lawrence C. McAfee、Michael Andersch、Mohammad Shoeybi 和 Bryan Catanzaro。减少大型Transformer模型中的激活值重计算。*ArXiv*，abs/2205.05198，2022年。

[20] Zhenzhong Lan、Mingda Chen、Sebastian Goodman、Kevin Gimpel、Piyush Sharma 和 Radu Soricut。ALBERT：用于语言表示自监督学习的轻量级 BERT。*arXiv* 预印本 *arXiv:1909.11942*，2019年。

[21] Chris Lattner、Mehdi Amini、Uday Bondhugula、Albert Cohen、Andy Davis、Jacques Pienaar、River Riddle、Tatiana Shpeisman、Nicolas Vasilache 和 Oleksandr Zinenko。MLIR：面向摩尔定律终结的编译器基础设施。*arXiv* 预印本 *arXiv:2002.11054*，2020年。

[22] Dmitry Lepikhin、HyoukJoong Lee、Yuanzhong Xu、Dehao Chen、Orhan Firat、Yanping Huang、Maxim Krikun、Noam Shazeer 和 Zhifeng Chen。GShard：通过条件计算和自动分片扩展巨型模型。*arXiv* 预印本 *arXiv:2006.16668*，2020年。

[23] Shen Li、Yanli Zhao、Rohan Varma、Omkar Salpekar、Pieter Noordhuis、Teng Li、Adam Paszke、Jeff Smith、Brian Vaughan、Pritam Damania等。Pytorch 分布式：加速数据并行训练的经验。*arXiv*预印本 *arXiv:2006.15704*，2020年。

[24] Zhuohan Li、Siyuan Zhuang、Shiyuan Guo、Danyang Zhuo、Hao Zhang、Dawn Song 和 Ion Stoica。TeraPipe：用于训练大规模语言模型的令牌级流水线并行。在国际机器学习会议，第 6543–6552 页。PMLR，2021年。

[25] Junyang Lin、Rui Men、An Yang、Chang Zhou、Ming Ding、Yichang Zhang、Peng Wang、Ang Wang、Le Jiang、Xianyan Jia 等。M6：中文多模态预训练模型。*arXiv* 预印本 *arXiv:2103.00823*，2021年。

[26] Yinhan Liu、Myle Ott、Naman Goyal、Jingfei Du、Mandar Joshi、Danqi Chen、Omer Levy、Mike Lewis、Luke Zettlemoyer和Veselin Stoyanov。Roberta：一种经过稳健优化的BERT预训练方法。*arXiv*预印本 *arXiv:1907.11692*，2019年。

[27] MPI论坛。MPI：消息传递接口标准，1994年。

[28] 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化流水线并行。在第*27*届*ACM*操作系统原理研讨会论文集，第1–15页，2019年。

[29] Deepak Narayanan、Mohammad Shoeybi、Jared Casper、Patrick LeGresley、Mostofa Patwary、Vijay Korthikanti、Dmitri Vainbrand、Prethvi Kashinkunti、Julie Bernauer、Bryan Catanzaro、Amar Phanishayee 和 Matei Zaharia。使用 Megatron-LM 在 GPU集群上进行高效的大规模语言模型训练。载于 高性能计算、网络、存储与分析国际会议论文集，SC '21，2021年。

[30] Adam Paszke、Sam Gross、Francisco Massa、Adam Lerer、James Bradbury、Gregory Chanan、Trevor Killeen、Zeming Lin、Natalia Gimelshein、Luca Antiga 等。PyTorch：一种命令式风格的高性能深度学习库。神经信息处理系统进展，32：8026–8037，2019年。

[31] Colin Raffel、Noam Shazeer、Adam Roberts、Katherine Lee、Sharan Narang、Michael Matena、Yanqi Zhou、Wei Li、Peter J Liu 等。使用统一的文本到文本 Transformer 探索迁移学习的极限。机器学习研究期刊，21(140)：1–67，2020年。

[32] Samyam Rajbhandari、Conglong Li、Zhewei Yao、Minjia Zhang、Reza Yazdani Aminabadi、Ammar Ahmad Awan、Jeff Rasley 和 Yuxiong He。DeepSpeed-MoE：推进混合专家推理和训练，助力下一代人工智能规模化发展。载于《国际机器学习会议》，第 18332–18346 页。PMLR，2022年。

[33] Samyam Rajbhandari、Jeff Rasley、Olatunji Ruwase 和 Yuxiong He。ZeRO：面向万亿参数模型训练的内存优化。载于 *SC20*：高性能计算、网络、存储与分析国际会议，第 1–16 页。IEEE，2020年。

[34] Jeff Rasley、Samyam Rajbhandari、Olatunji Ruwase 和 Yuxiong He。DeepSpeed：系统优化使得训练参数量超过 1000 亿的深度学习模型成为可能。载于 第 *26* 届 *ACM SIGKDD*国际知识发现与数据挖掘大会论文集，第 3505–3506 页，2020年。

[35] Jared Roesch、Steven Lyubomirsky、Marisa Kirisame、Logan Weber、Josh Pollock、Luis Vega、Ziheng Jiang、Tianqi Chen、Thierry Moreau 和 Zachary Tatlock。《Re- lay：面向深度学习的高级编译器》。*arXiv*预印本 *arXiv:1904.08368*，2019年。

[36] Robin Rombach、Andreas Blattmann、Dominik Lorenz、Patrick Esser 和 Björn Ommer。《利用潜在扩散模型进行高分辨率图像合成》。载于*IEEE/CVF*计算机视觉与模式识别会议论文集，第10684–10695页，2022年。

[37] Olaf Ronneberger、Philipp Fischer 和 Thomas Brox。U-Net：用于生物医学图像分割的卷积网络。发表于医学图像计算与计算机辅助干预国际会议，第 234–241 页。施普林格，2015年。[38] Alexander Sergeev 和 Mike Del Balso。Horovod：在 TensorFlow 中快速轻松地进行分布式深度学习。*arXiv* 预印本 *arXiv:1802.05799*，2018年。[39] Noam Shazeer、Youlong Cheng、Niki Parmar、Dustin Tran、Ashish Vaswani、Penporn Koanantakool、Peter Hawkins、HyoukJoong Lee、Mingsheng Hong、Cliff Young 等。Mesh-TensorFlow：面向超级计算机的深度学习。*arXiv* 预印本 *arXiv:1811.02084*，2018年。[40] Noam Shazeer、Azalia Mirhoseini、Krzysztof Maziarz、Andy Davis、Quoc Le、Geoffrey Hinton 和 Jeff Dean。规模惊人的大型神经网络：稀疏门控专家混合层。*arXiv* 预印本 *arXiv:1701.06538*，2017年。[41] Karen Simonyan 和 Andrew Zisserman。用于大规模图像识别的超深卷积网络。*arXiv* 预印本 *arXiv:1409.1556*，2014年。[42] Shaden Smith、Mostofa Patwary、Brandon Norick、Patrick LeGresley、Samyam Rajbhandari、Jared Casper、Zhun Liu、Shrimai Prabhumoye、George Zerveas、Vijay Korthikanti 等。使用 DeepSpeed 和 Megatron 训练 Megatron-Turing NLG 530B：大型生成式语言模型。*arXiv* 预印本 *arXiv:2201.11990*，2022年。[43] 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高性能国际研讨会计算机体系结构 *(HPCA)*，第 342–355 页。IEEE，2020年。

[44] Colin Unger、Zhihao Jia、Wei Wu、Sina Lin、Mandeep Baines、Carlos Efrain Quintero Narvaez、Vinay Ramakr- ishnaiah、Nirmal Prajapati、Pat McCormick、Jamaludin Mohd-Yusof 等。《Unity：通过联合优化代数变换与并行化，加速 {DNN} 训练》。载于第*16*届 *USENIX* 操作系统设计与实现研讨会（*OSDI 22*），第267–284页，2022年。

[45] Minjie Wang、Chien-chin Huang 和 Jinyang Li。使用自动数据流图划分支持超大型模型。载于第十四届 *EuroSys* 会议论文集，*2019*年，第 1–17 页，2019年。

[46] XLA 和 TensorFlow 团队。Xla - tensorflow，已编译。2017年。

[47] Yuanzhong Xu、HyoukJoong Lee、Dehao Chen、Blake Hechtman、Yanping Huang、Rahul Joshi、Maxim Krikun、Dmitry Lepikhin、Andy Ly、Marcello Maggioni 等。GSPMD：面向机器学习计算图的通用且可扩展的并行化。*arXiv* 预印本 *arXiv:2105.04663*，2021年。[48] Sergey Zagoruyko 和 Nikos Komodakis。宽残差网络。*arXiv* 预印本 *arXiv:1605.07146*，2016年。[49] Lianmin Zheng、Zhuohan Li、Hao Zhang、Yonghao Zhuang、Zhifeng Chen、Yanping Huang、Yida Wang、Yuanzhong Xu、Danyang Zhuo、Joseph E Gonzalez 等。Alpa：自动化分布式深度学习的算子间和算子内并行化。*arXiv* 预印本 *arXiv:2201.12023*，2022年。