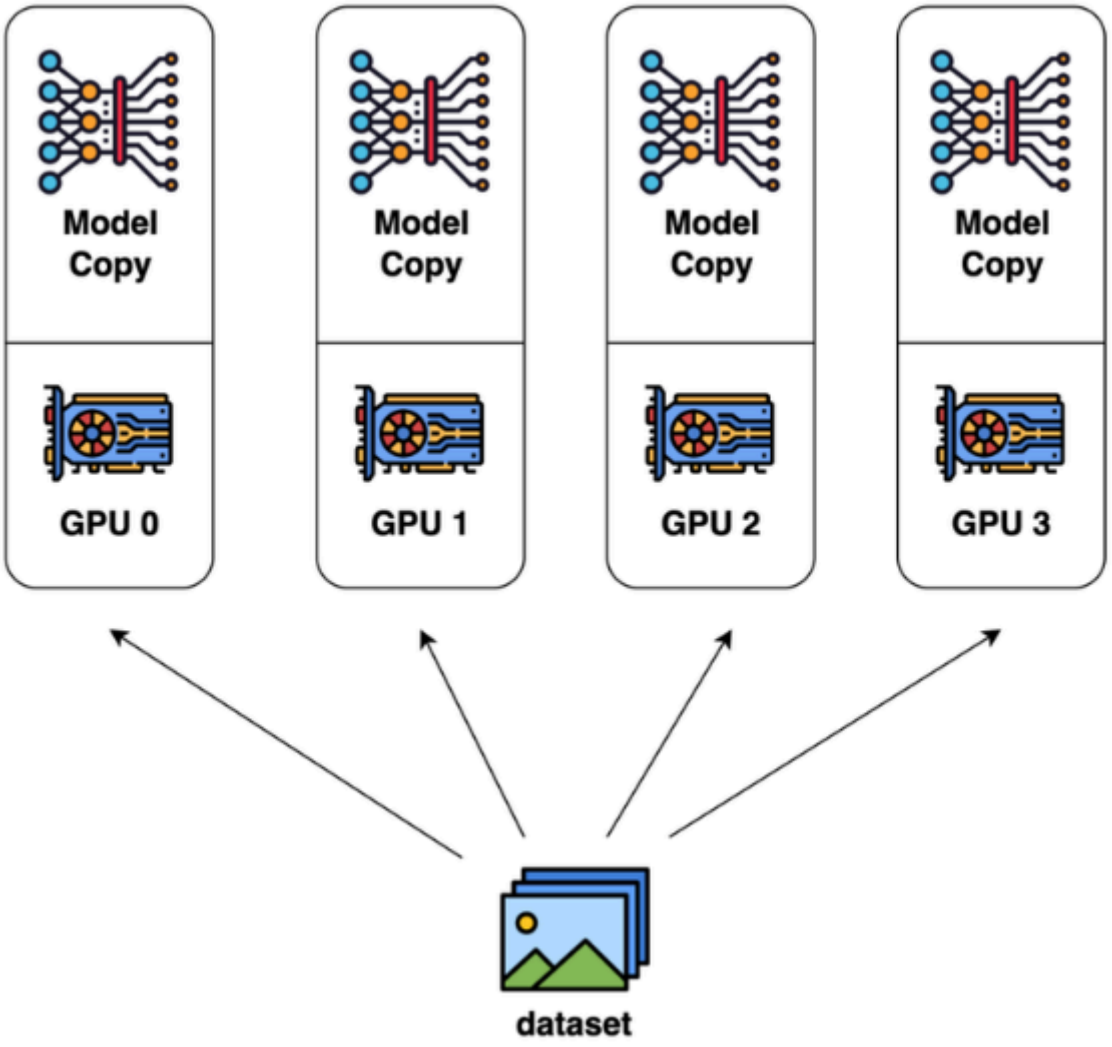
1.概述

1.数据并行

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，数据集被分割成几个碎片，每个碎片被分配到一个设备上。这相当于**沿批次（Batch）维度对训练过程进行并行化。数据并行是将一个大批次（batch）拆分成多个小批次，分别发送到不同的设备（如 GPU）上并行计算，最后将结果合并。**每个设备将持有一个完整的模型副本，并在分配的数据集碎片上进行训练。在反向传播之后，模型的梯度将被全部减少，以便在不同设备上的模型参数能够保持同步。典型的数据并行实现：PyTorch DDP。



> **Note**
>
> PyTorch 的 `DistributedDataParallel (DDP)` 是一种高效的数据并行实现方式：
>
> - 它不会像老版本的 `DataParallel` 那样只在单机多卡中使用主卡聚合梯度。而是通过分布式通信（如 NCCL、Gloo 后端）让所有设备**对等地通信梯度**，使用 AllReduce 技术，所有设备平等地交换梯度信息。
> - 使用 `torch.distributed` 初始化进程组，每个设备独立运行一部分数据。在训练过程中，每个设备都是一个独立的进程，拥有完整的模型副本和属于自己的一部分数据，从而实现真正意义上的分布式训练。

2.模型并行

在数据并行训练中，一个明显的特点是每个 GPU 持有整个模型权重的副本。这就带来了冗余问题。另一种并行模式是**模型并行，即模型被分割并分布在一个设备阵列上。**

通常有两种类型的模型并行：**张量并行**和**流水线并行。**

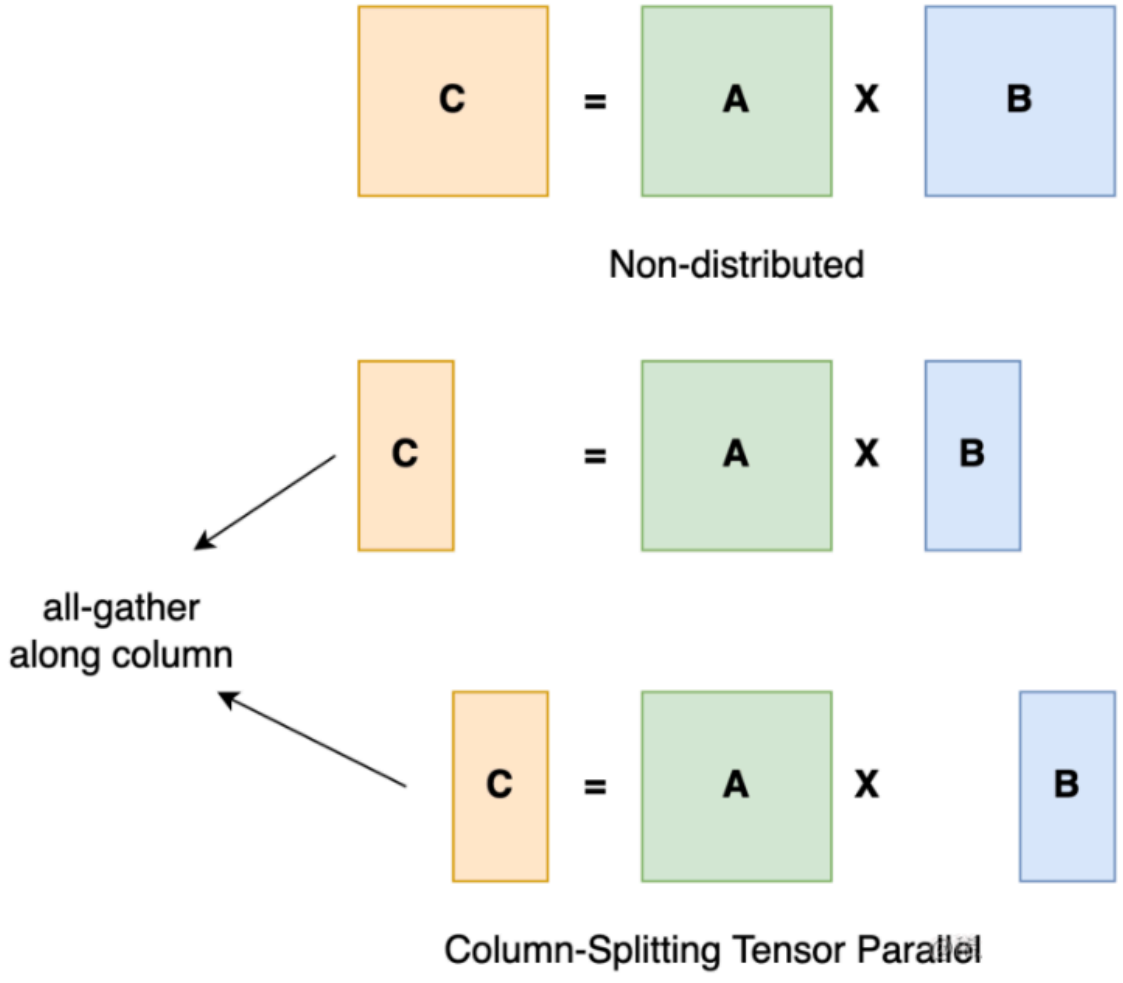
- **张量并行是在一个操作中进行并行计算**，如：矩阵-矩阵乘法。
- **流水线并行是在各层之间进行并行计算。**

因此，从另一个角度来看，**张量并行可以被看作是层内并行，流水线并行可以被看作是层间并行。**

2.1 张量并行

张量并行训练是**将一个张量沿特定维度分成 N 块，每个设备只持有整个张量的 1/N，同时不影响计算图的正确性。**这需要额外的通信来确保结果的正确性。

以一般的矩阵乘法为例，假设我们有 `C = AB`。我们可以将B沿着列分割成 `[B0 B1 B2 ... Bn]`，每个设备持有一列。然后我们将 A 与每个设备上 B 中的每一列相乘，我们将得到 `[AB0 AB1 AB2 ... ABn]`。此刻，每个设备仍然持有一部分的结果，例如，设备(rank=0)持有 AB0。为了确保结果的正确性，我们需要收集全部的结果，并沿列维串联张量。通过这种方式，我们能够将张量分布在设备上，同时确保计算流程保持正确。



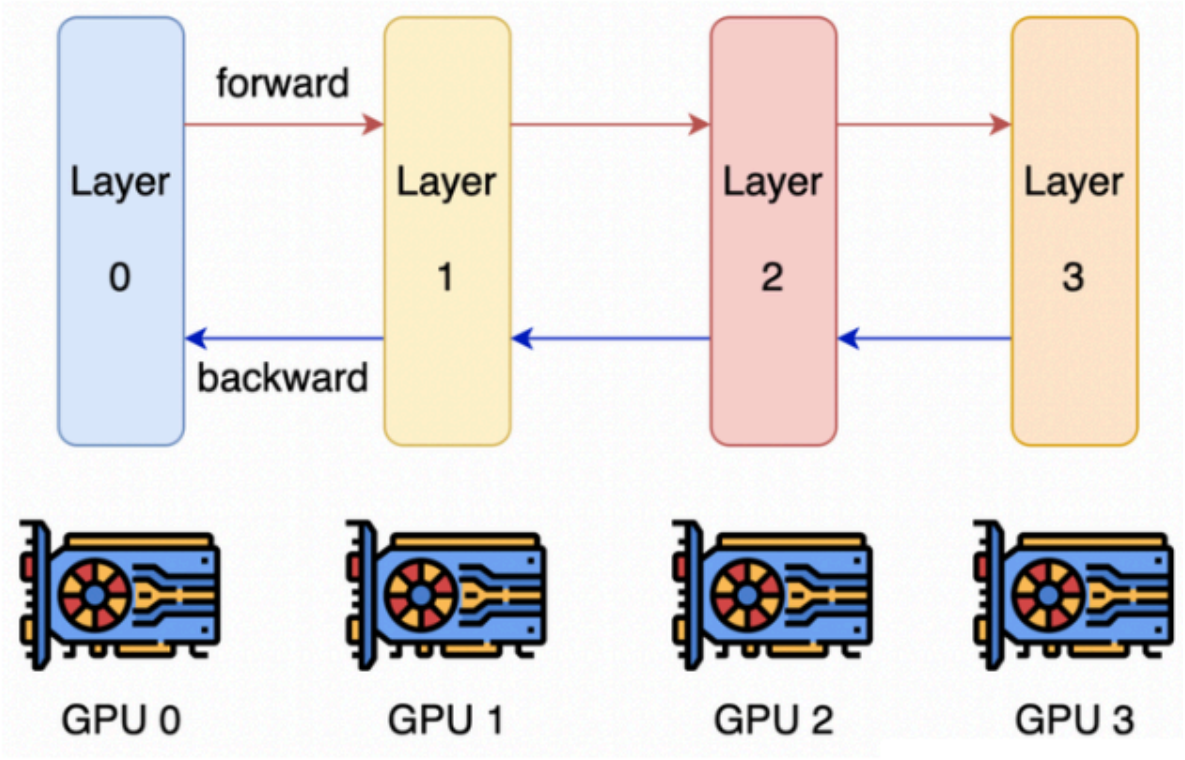
典型的张量并行实现：Megatron-LM（1D）、Colossal-AI（2D、2.5D、3D）。

2.2 流水线并行

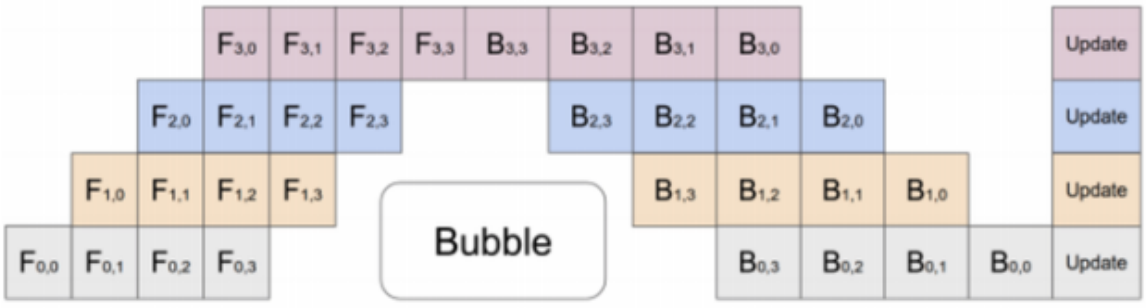
流水线并行的核心思想是，**模型按层分割成若干块，每块都交给一个设备。**

- 在前向传播过程中，每个设备将中间的激活传递给下一个阶段。
- 在后向传播过程中，每个设备将输入张量的梯度传回给前一个流水线阶段。

这允许设备同时进行计算，从而增加训练的吞吐量。



流水线并行训练的一个明显**缺点是训练设备容易出现空闲状态**（因为后一个阶段需要等待前一个阶段执行完毕），导致计算资源的浪费，加速效率没有数据并行高。

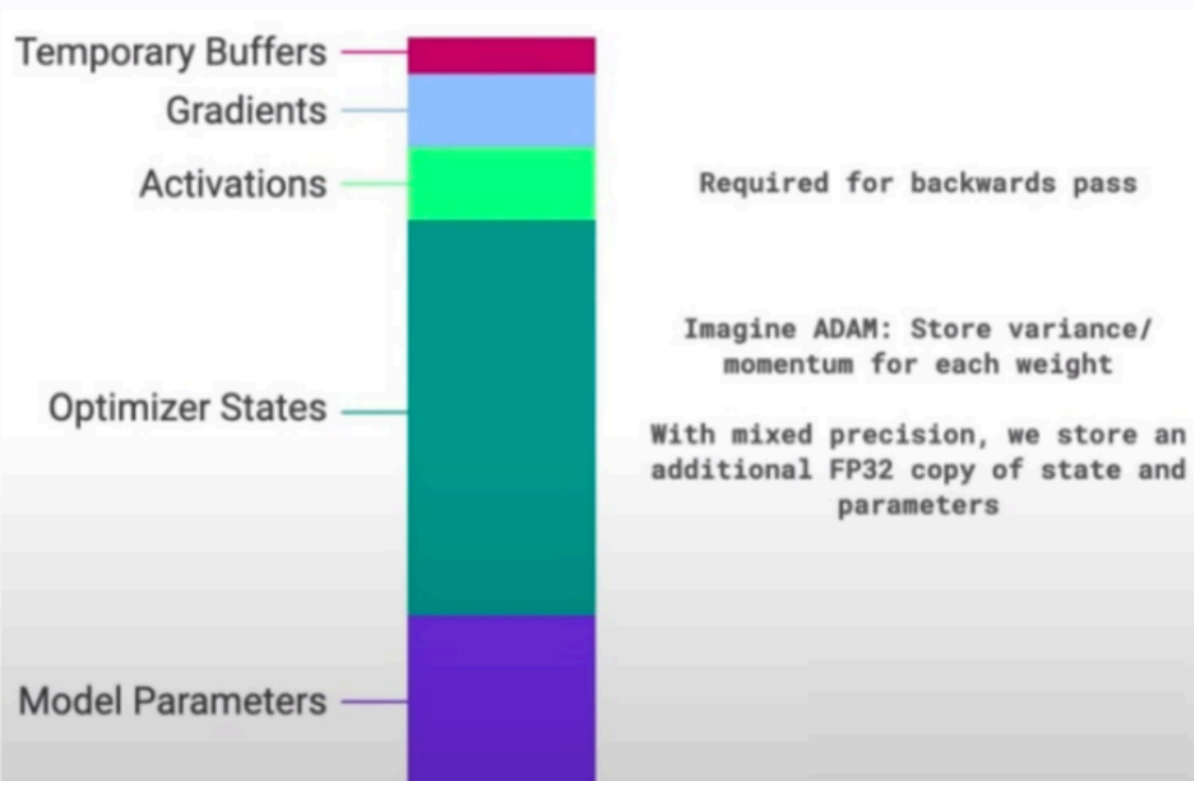


典型的流水线并行实现：GPipe、PipeDream、PipeDream-2BW、PipeDream Flush（1F1B）。

3.优化器相关的并行

目前随着模型越来越大，**单个GPU的显存目前通常无法装下那么大的模型了。**那么就要想办法对占显存的地方进行优化。

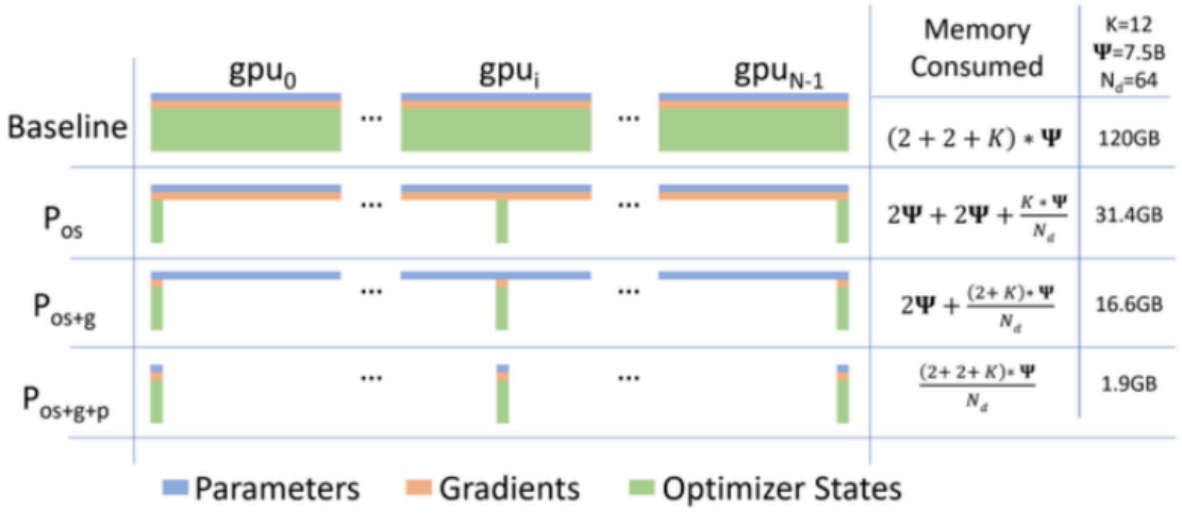
通常来说，模型训练的过程中，GPU上需要进行存储的参数包括了模型本身的参数、优化器状态、激活函数的输出值、梯度以及一些零时的Buffer。各种数据的占比如下图所示：



可以看到模型参数仅占模型训练过程中所有数据的一部分，当进行混合精度运算时，其中模型状态参数(优化器状态 + 梯度+ 模型参数）占到了一大半以上。因此，我们**需要想办法去除模型训练过程中的冗余数据。**

而优化器相关的并行就是一种去除冗余数据的并行方案，目前这种并行最流行的方法是 **ZeRO**（即零冗余优化器）。针对模型状态的存储优化（去除冗余），**ZeRO使用的方法是分片，即每张卡只存 1/N 的模型状态量，这样系统内只维护一份模型状态。**ZeRO有三个不同级别，对模型状态进行不同程度的分片：

- ZeRO-1 : 对优化器状态分片（Optimizer States Sharding）
- ZeRO-2 : 对优化器状态和梯度分片（Optimizer States & Gradients Sharding）
- ZeRO-3 : 对优化器状态、梯度分片以及模型权重参数分片（Optimizer States & Gradients & Parameters Sharding）

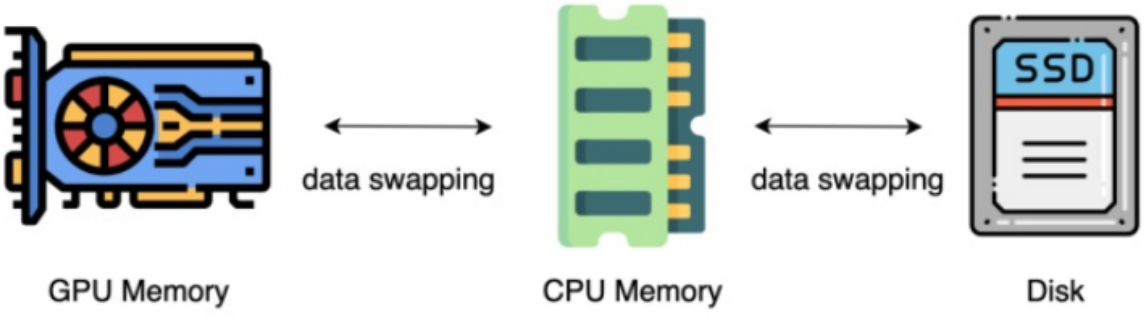


4.异构系统并行

上述的方法中，通常需要大量的 GPU 来训练一个大型模型。然而，人们常常忽略一点，与 GPU 相比，CPU 的内存要大得多。在一个典型的服务器上，CPU 可以轻松拥有几百GB甚至上TB的内存，而每张 GPU 卡通常只有 48 或 80 GB的内存。这促使人们思考为什么 CPU 内存没有被用于分布式训练。

而最近的进展是依靠 CPU 甚至是 NVMe 磁盘来训练大型模型。主要的想法是，**在不使用张量时，将其卸载回 CPU 内存或 NVMe 磁盘。**

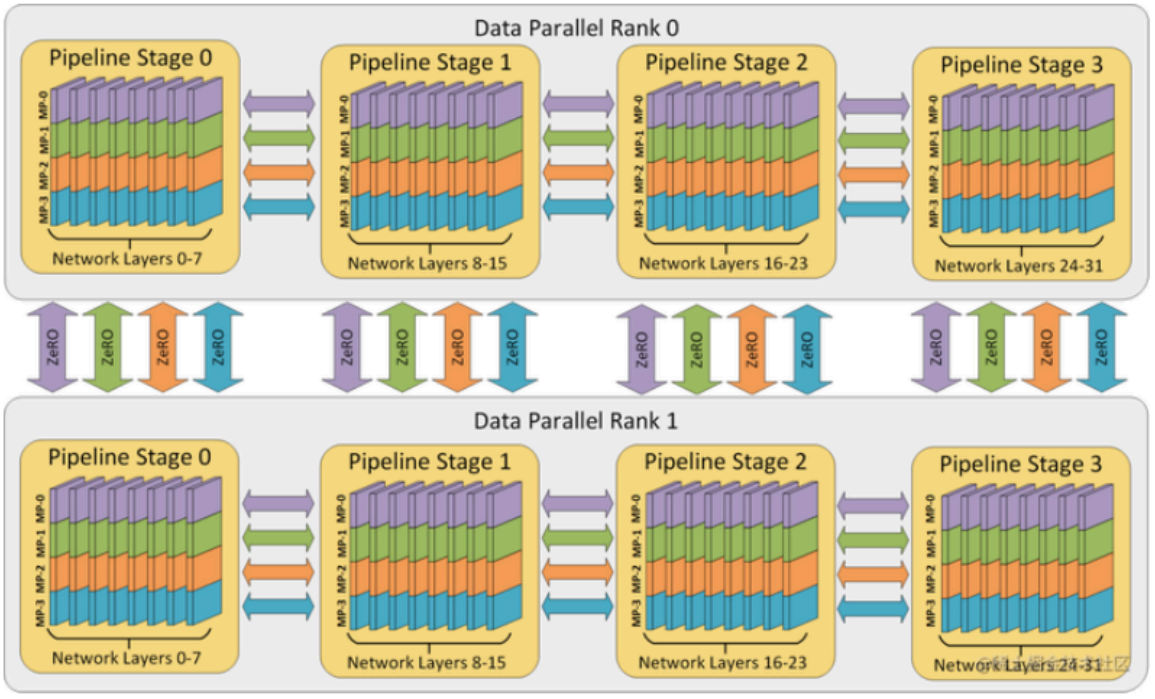
通过使用异构系统架构，有可能在一台机器上容纳一个巨大的模型。



5.多维混合并行

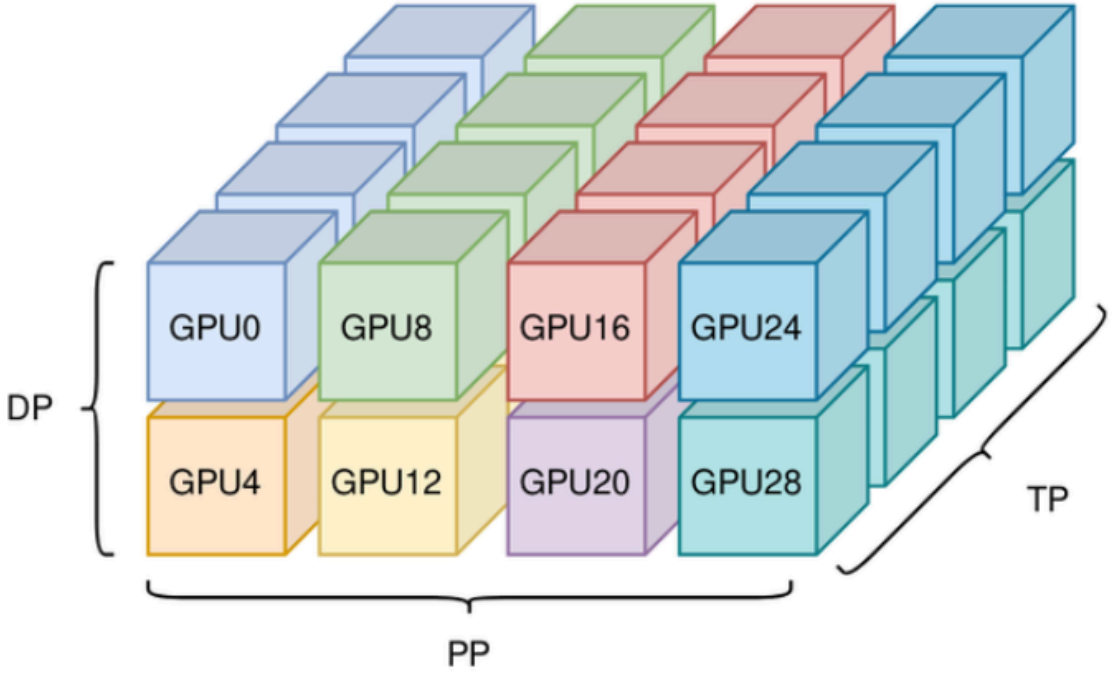
多维混合并行指将数据并行、模型并行和流水线并行等多种并行技术结合起来进行分布式训练。

数据并行+张量并行+流水线并行：

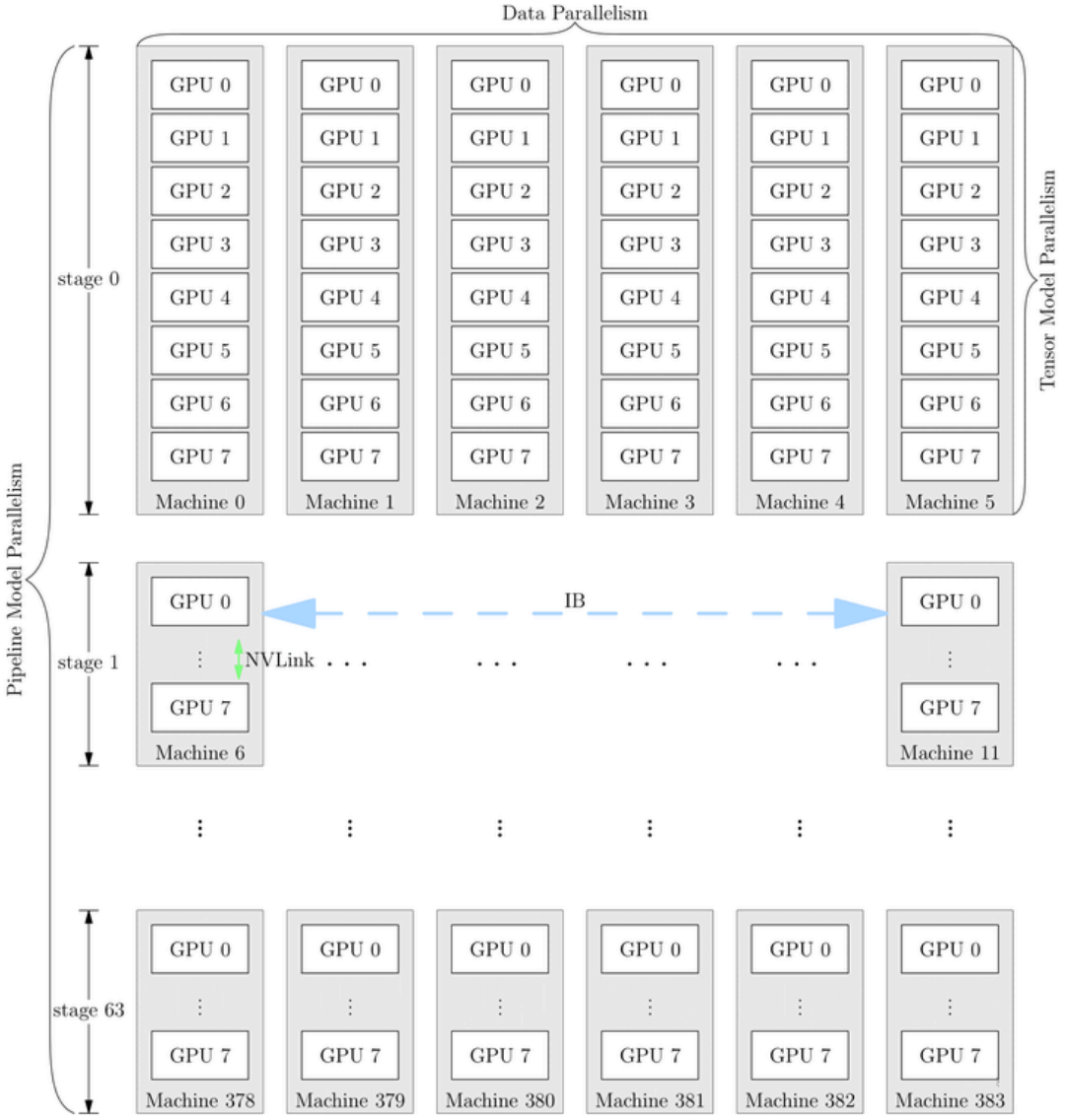


通常，在进行超大规模模型的预训练和全参数微调时，都需要用到多维混合并行：

1. **数据并行（Data Parallelism, DP）**
2. **张量并行（Tensor Parallelism, TP）**
3. **流水线并行（Pipeline Parallelism, PP）**



为了充分利用带宽，通常情况下，张量并行所需的通信量最大，而数据并行与流水线并行所需的通信量相对来说较小。因此，**同一个服务器内使用张量并行，而服务器之间使用数据并行与流水线并行。**



6.自动并行

上面提到的数据并行、张量并行、流水线并行等多维混合并行需要把模型切分到多张AI加速卡上面，如果让用户手动实现，对开发者来说难度非常大，需要考虑性能、内存、通信、训练效果等问题，要是能够将模型按算子或者按层自动切分到不同的加速卡上，可以大大的降低开发者的使用难度。因此，自动并行应运而生。

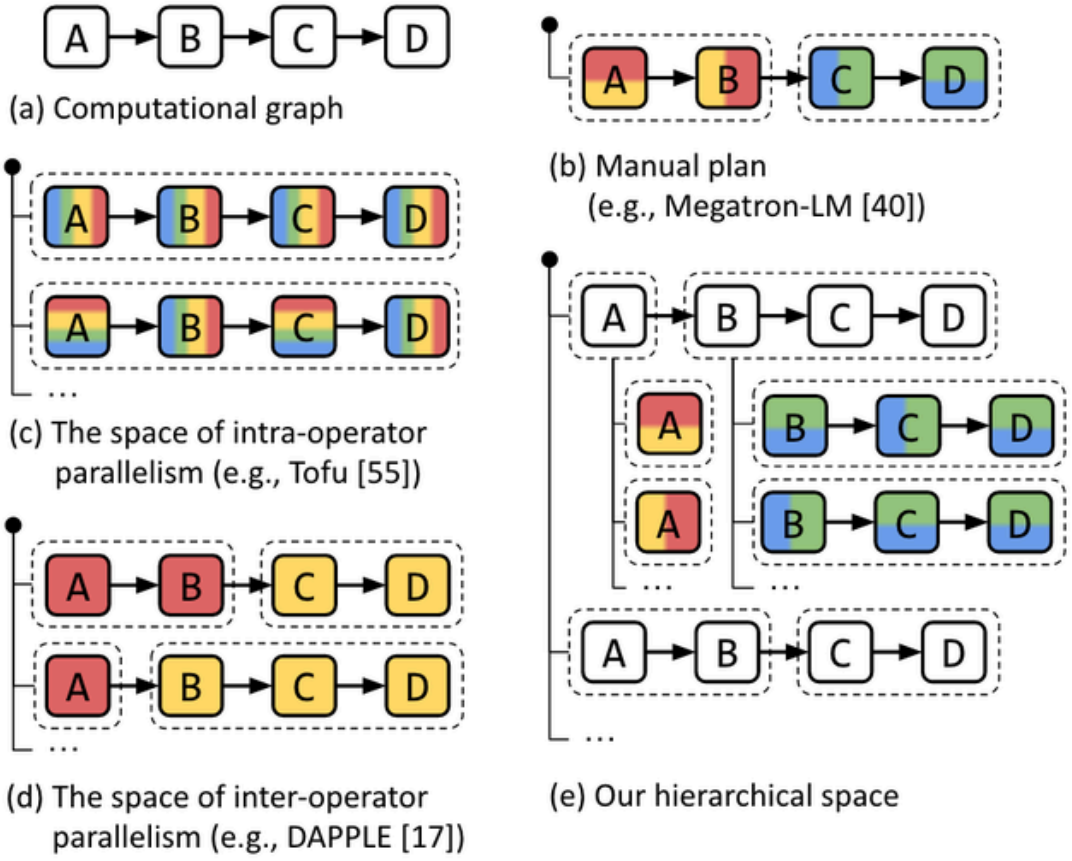


Figure 1: Generation of parallelization plans for a computational graph shown in (a). Different colors represent different devices, dashed boxes represent pipeline stages. (b) creates the plan manually. (c) and (d) automatically generate plans using only one of intra- and inter-operator parallelisms. (e) shows our approach that creates a hierarchical space to combine intra- and inter-operator parallelisms.

图中关键点总结：

部分	描述	特点
(a)	原始计算图	无并行化，按顺序执行
(b)	手动并行化计划	手动划分流水线阶段，灵活性高
(c)	层内并行（Intra-Operator Parallelism）	自动拆分节点到多个设备，适合张量并行

部分	描述	特点
(d)	层间并行（Inter-Operator Parallelism）	自动跨设备分配节点，适合数据并行或流水线并行
(e)	层次化并行空间	结合层内并行和层间并行，灵活性更高

7.MOE并行/专家并行

通常来讲，模型规模的扩展会导致训练成本显著增加，计算资源的限制成为了大规模密集模型训练的瓶颈。为了解决这个问题，一种基于稀疏 MoE 层的深度学习模型架构被提出，即**将大模型拆分成多个小模型(专家，`expert`)，每轮迭代根据样本决定激活一部分专家用于计算，达到了节省计算资源的效果**；并引入可训练并确保稀疏性的门(`gate`)机制，以保证计算能力的优化。

使用 MoE 结构，可以在计算成本次线性增加的同时实现超大规模模型训练，为恒定的计算资源预算带来巨大增益。而 **MOE 并行，本质上也是一种模型并行方法。**下图展示了一个有六个专家网络的模型被两路专家并行地训练。其中，专家1-3被放置在第一个计算单元上，而专家4-6被放置在第二个计算单元上。