调研报告

杭州海康威视数字技术股份有限公司

文档编号

GPU并行化计算调研报告

|  |  |
| --- | --- |
| 编制 | 万骏泓 |
| 审批 |  |

密级级别：[内部公开]

生效时间： 2020年2月19日

保密期：无

杭州海康威视数字技术股份有限公司 版权所有

**目录**

[1. 简介 3](#_Toc33081513)

[1.1 编写目的 3](#_Toc33081514)

[1.2 研究背景及意义 3](#_Toc33081515)

[2. 绪论 3](#_Toc33081516)

[3. 底层 3](#_Toc33081517)

[3.1 通信框架 4](#_Toc33081518)

[3.1.1 MPI 4](#_Toc33081519)

[3.1.2 NCCL 4](#_Toc33081520)

[3.2 通信模式 4](#_Toc33081521)

[3.3 通信构架 5](#_Toc33081522)

[3.3.1 Ring-base collectives 6](#_Toc33081523)

[4. 多GPU构架 7](#_Toc33081524)

[5. 模型训练 8](#_Toc33081525)

[5.1 数据并行 9](#_Toc33081526)

[5.1.1 同步更新 9](#_Toc33081527)

[5.1.2 异步更新 10](#_Toc33081528)

[5.1.3 Allreduce模式 11](#_Toc33081529)

[5.1.4 梯度平均 11](#_Toc33081530)

[5.2 模型并行 12](#_Toc33081531)

[5.3 比较 12](#_Toc33081532)

[6. 修订记录 13](#_Toc33081533)

# 简介

## 编写目的

应对目前越来越大规模的数据以及越来越复杂的模型对时间成本和设备硬件的提出的挑战，多GPU并行化计算取得了显著的进展。本文从底层通信，框架和模型几方面入手，总结整理目前流行并大规模运用的多GPU并行化计算方案。

## 研究背景及意义

随着深度学习算法在工业界也取得显著效果同时，深度学习庞大的参数量，复杂的模型构架以及训练时需要更大量的数据和时间，对设备硬件提出了更高的要求。而多GPU并行化计算正是解决这种问题的目前最好的解决方案。

# 绪论

深度学习中常常需要多GPU并行训练，在增大数据集的同时增大模型参数量是提高准确率的一个有效方案。但这也意味着计算量和训练时间的快速上升。为了缩减训练时间，我们可以使用分布式并行训练。

搭建一个大型的分布式系统是一个耗时耗力的大工程。在量级不那么大的训练场景下，通常多卡并行是一个简单且高效的方案。单机多卡系统可以认为是分布式系统的一种简单特例：卡间通信走PCIe (或者Nvlink)，要比多机之间走以太网(Ethernet)快很多。

本文将简单的介绍多GPU并行化计算的底层构架，常用的通信方式，最后将介绍针对不同需求的模型训练并行化方案。

# 底层

首先多GPU需要通过一定的方式连接在一起，并且互相之间需要信息的交互，这又提出了不同的通信手段和方案。 而每个主机上都可以有多个设备(可以是 GPU 或 CPU)。主机和设备通常抽象成计算节点(node)，可以进行运算以及和其他节点通信。

## 通信框架

### MPI

在计算时 节点 间通常需要相互通信。MPI (message passing interface) 是一个跨语言的通讯协议，用于编写并行计算机。MPI是一个信息传递应用程序接口，包括协议和和语义说明，他们指明其如何在各种实现中发挥其特性。MPI的目标是高性能，大规模性，和可移植性。MPI在今天仍为高性能计算的主要模型。

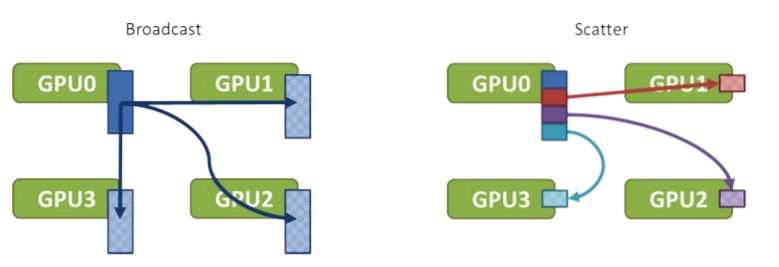
### NCCL

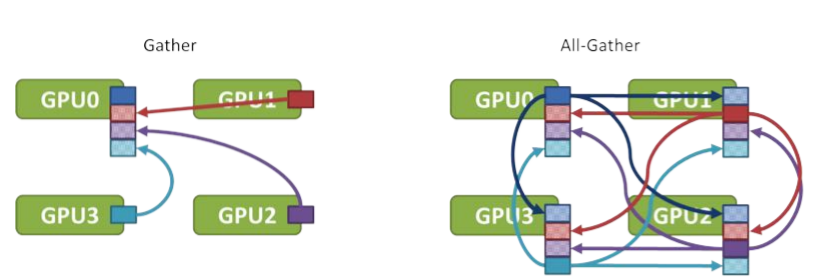
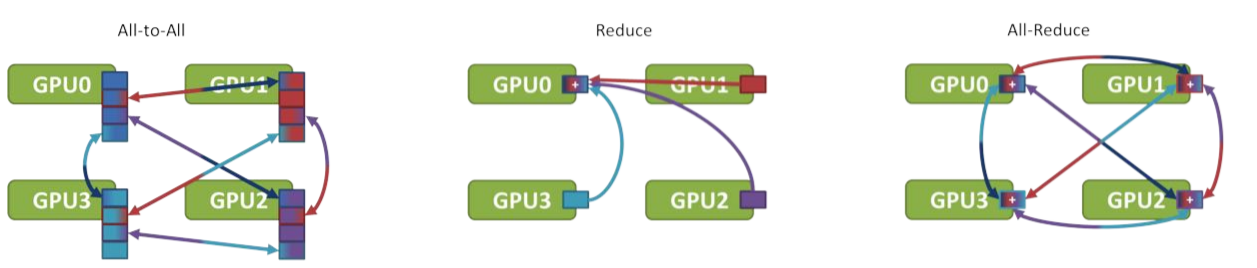
虽然MPI在并行计算的成功，但是MPI 是针对大规模的节点之间的并行化计算。随着目前深度学习的发展，多GPU并行化计算通常是在小规模的节点之间（通常为8卡或者4卡），但是深度学习模型需要多节点之间有更多的通信宽带以及更效率的通信手段。这这种需求下，Nvdia开发了NCCL专门针对深度学习的需要。

NCCL是Nvidia Collective multi-GPU Communication Library的简称，它是一个实现多GPU的collective communication通信（all-gather, reduce, broadcast）库，Nvidia做了很多优化，以在PCIe、Nvlink、InfiniBand上实现较高的通信速度。深度学习中常常需要多GPU并行训练，而Nvidia的NCCL库在各大深度学习框架（Caffe/Tensorflow/Torch/Theano）的多卡并行中经常被使用，下文主要为基于NCCL的通信模式，通信构架介绍。

## 通信模式

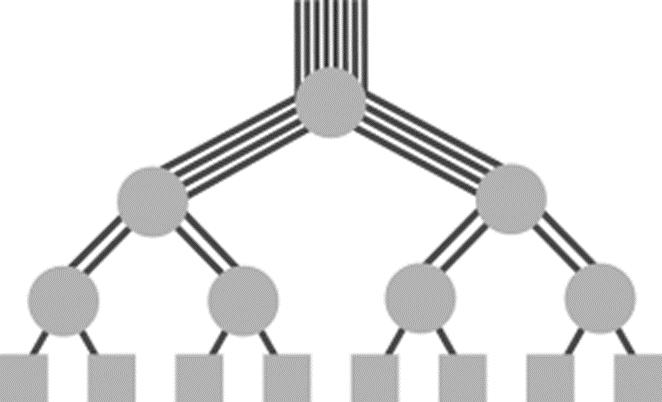
并行任务的通信一般可以分为Point-to-point communication和Collective communication。 P2P通信这种模式只有一个发送者和一个接收者，实现起来比较简单。第二种Collective communication包含多个发送者多个接收者，也是目前主流多GPU并行化计算所用的通信模式。一般的通信原语包括broadcast，gather, all-gather, scatter, reduce, all-reduce, reduce-scatter, all-to-all等，具体见下图。



## 通信构架

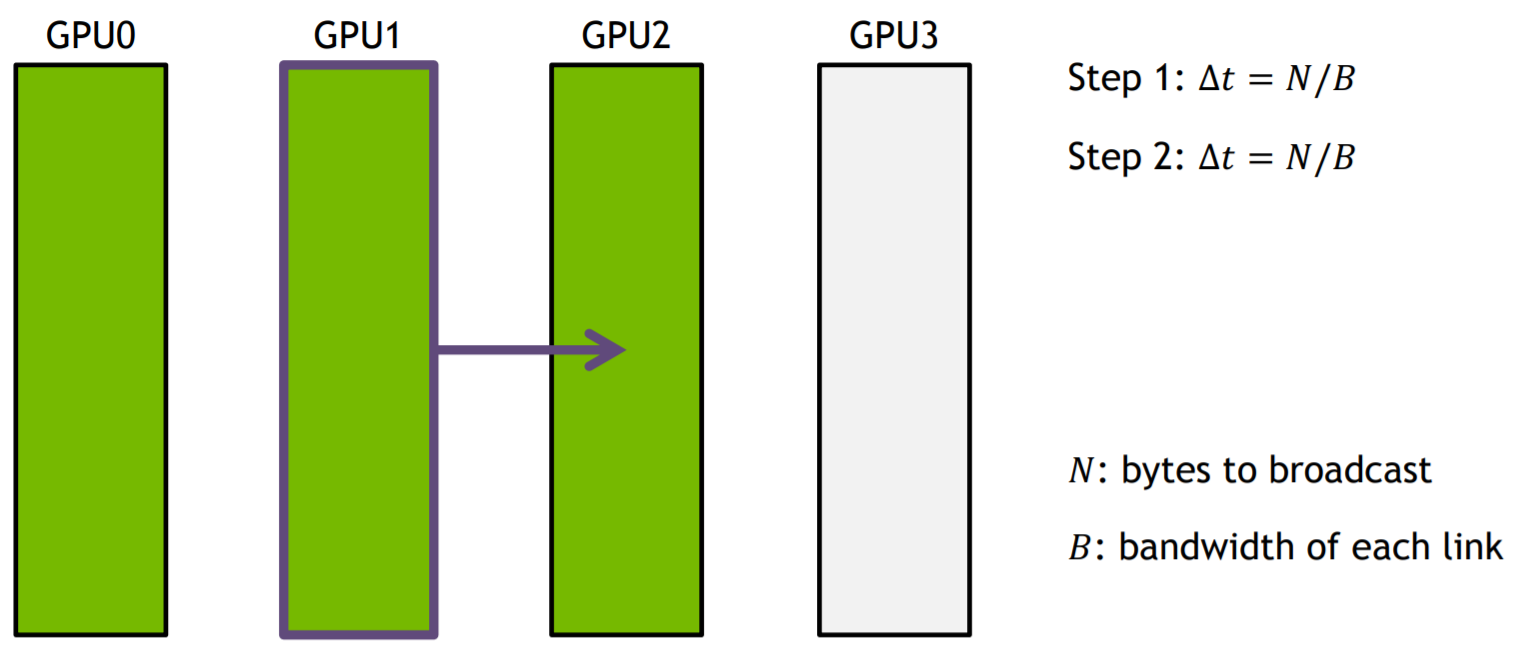
传统collective communication假设通信节点组成的构架是一颗fat tree，如下图所示，这样通信效率最高。



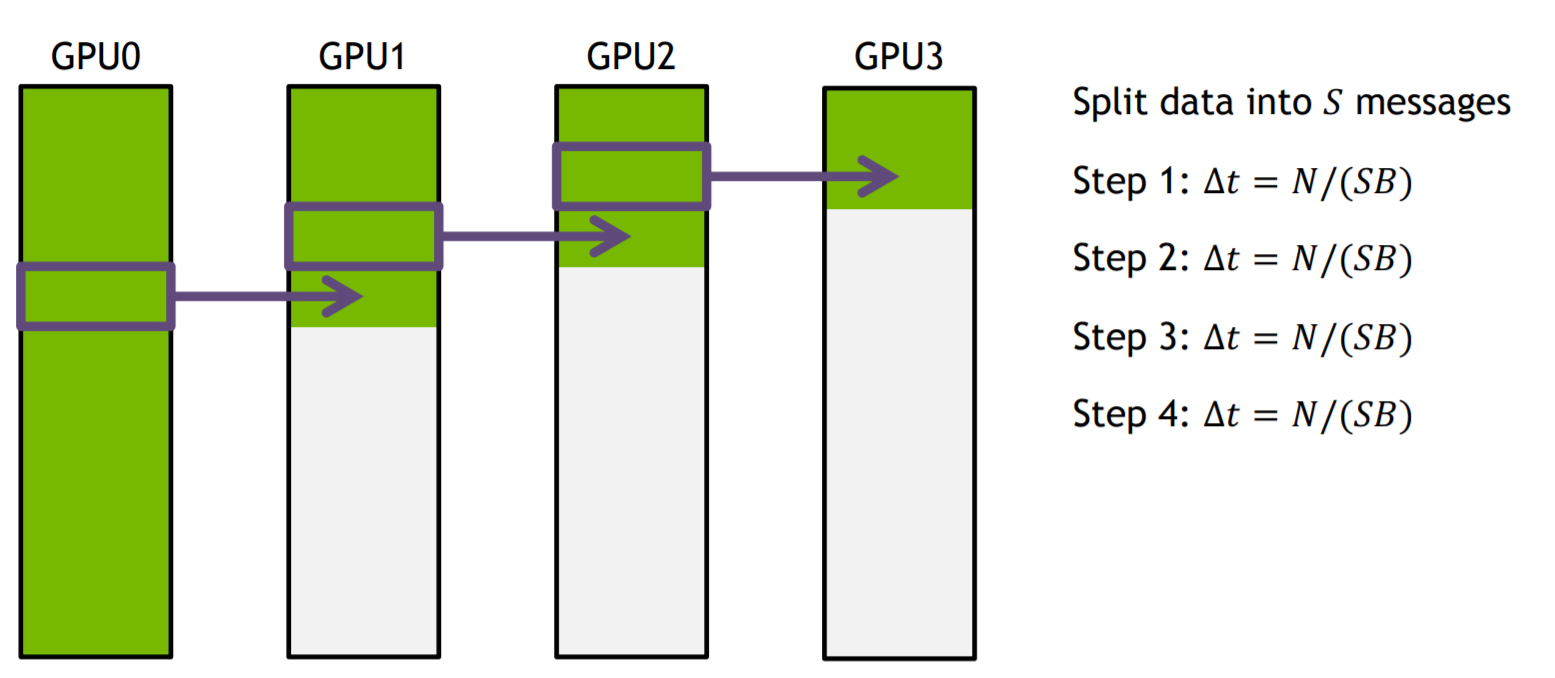
但实际的通信构架可能比较复杂，并且可以看出每个节点需要承担的通信任务量是不同的，这就可能导致由于某些节点较慢拖累整体并行化计算的效率的问题。因此一般用ring-based Collective communication。

### Ring-base collectives

Ring-base collectives将所有的通信节点通过首尾连接形成一个单向环，数据在环上依次传输。以broadcast为例， 假设有4个GPU，GPU0为发送者将信息发送给剩下的GPU，按照环的方式依次传输，GPU0-->GPU1-->GPU2-->GPU3，若数据量为N，带宽为B，K个设备，整个传输时间为（K-1）N/B。时间随着节点数线性增长，不是很高效。

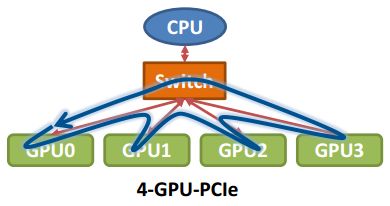


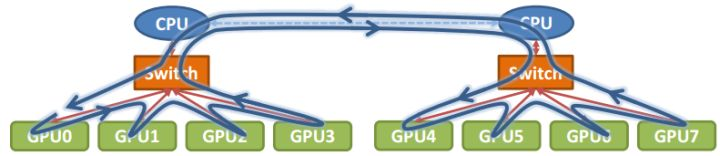
下面把要传输的数据分成S份，每次只传N/S的数据量，GPU1接收到GPU0的一份数据后，也接着传到环的下个节点，这样以此类推，最后花的时间为S\*(N/S/B) + (k-2)\*(N/S/B) = N(S+K-2)/(SB) --> N/B，条件是S远大于K，即数据的份数大于节点数，这个很容易满足。所以通信时间不随节点数的增加而增加，只和数据总量以及带宽有关。其它通信操作比如reduce、gather以此类推。



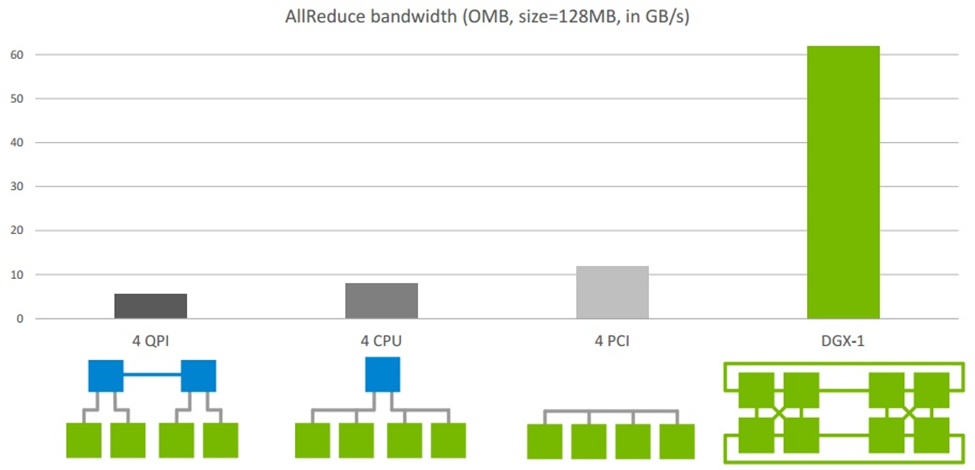
# 多GPU构架

多GPU通过什么样的方式，互相连接也决定了互相通信，传递数据的效率。我们传统的单机多卡通信环的组成，一般是单机4卡通过同一个PCIe switch挂载在一棵CPU的场景，或者是两机8卡通过两个CPU下不同的PCIe switch挂载的场景。



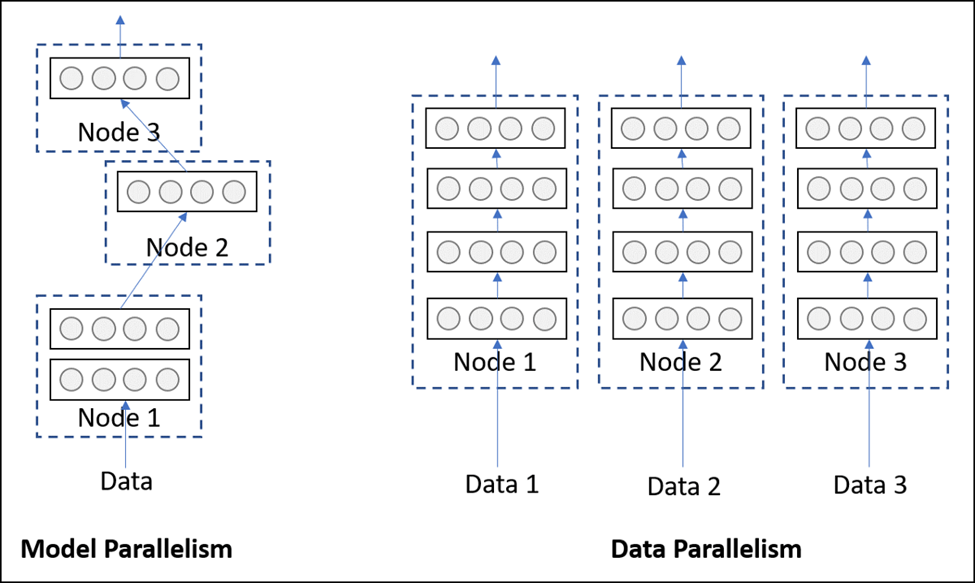


多卡的构架方式不同也会导致互相之间的通信效率的显著差异。DGX-1架构是Nvidia推出的深度学习平台，带宽能达到60GB/s。前面三个是单机多卡典型的三种连接方式，第三种是四张卡都在一个PCIe switch上，所以带宽较高，能达到>10GB/s PCIe的带宽大小，第二种是两个GPU通过switch相连后再经过CPU连接，速度会稍微低一点，第一种是两个GPU通过CPU然后通过QPI和另一个CPU上的两块卡相连，因此速度最慢，但也能达到>5GB/s。



# 模型训练

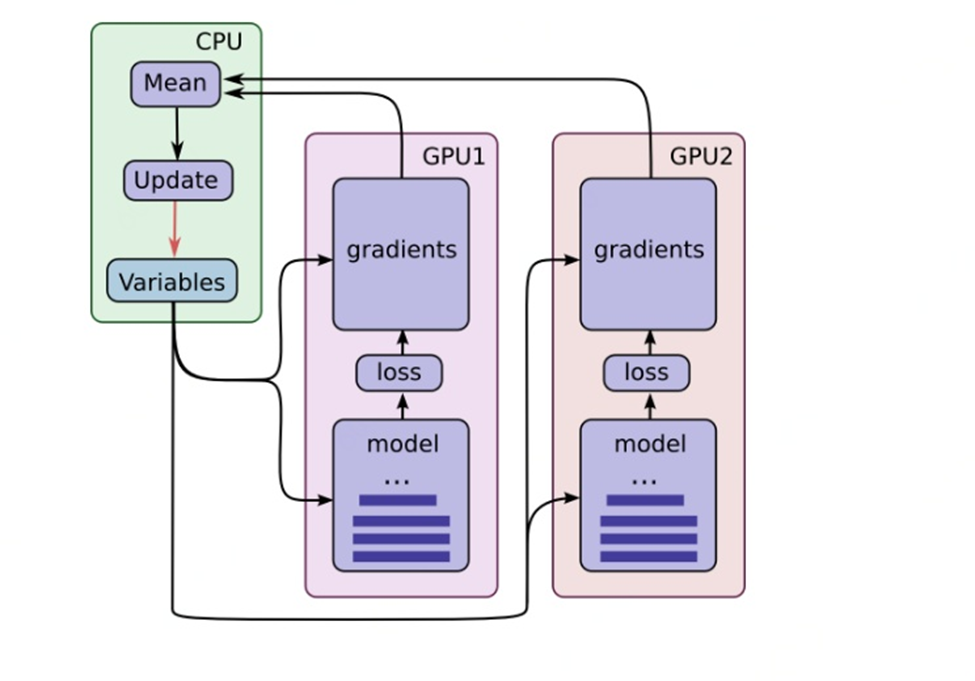
深度学习模型的并行有两种方案：模型并行(model parallel)和数据并行(data parallel)。模型并行：不同 node 输入相同数据，运行模型的不同部分；数据并行：不同 node 输入不同数据，运行相同的完整的模型。



为了完成一次更新，不同节点间需要交换正向传播和反向传播的信息，所以通信数据量是选择这两种并行方案的一个考量因素。另一个考量因素是由数据依赖带来的计算的阻塞。最后一个考量因素是内存限制。当模型参数以及计算产生的中间变量无法放入一个节点的内存时，我们只能使用模型并行。当参数量巨大的时候，数据通信量会成为模型运行的瓶颈。模型并行的数据通信量可以比数据并行更少。

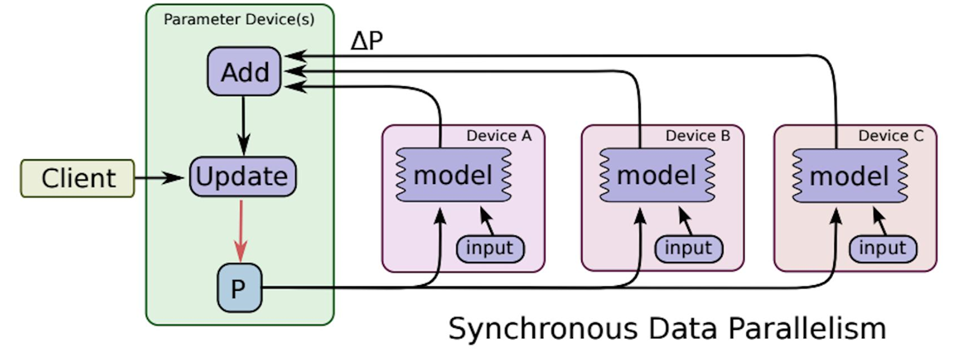
## 数据并行

数据并行是大规模运用的多GPU计算方案，主要特别就是能大幅度优化训练时间。数据并行是在每个节点上用相同的模型，通过mini-batch的形式，将数据划分切割为k份，分别发送给k个节点中。每个节点计算后得出的梯度信息统一上传到到统一的parameter server（ps通常为CPU）进行梯度下降，ps 拿到参数后，视更新的方案不同可以分为同步更新 (synchronous update) 和异步更新 (asynchronous update)，以此循环迭代。



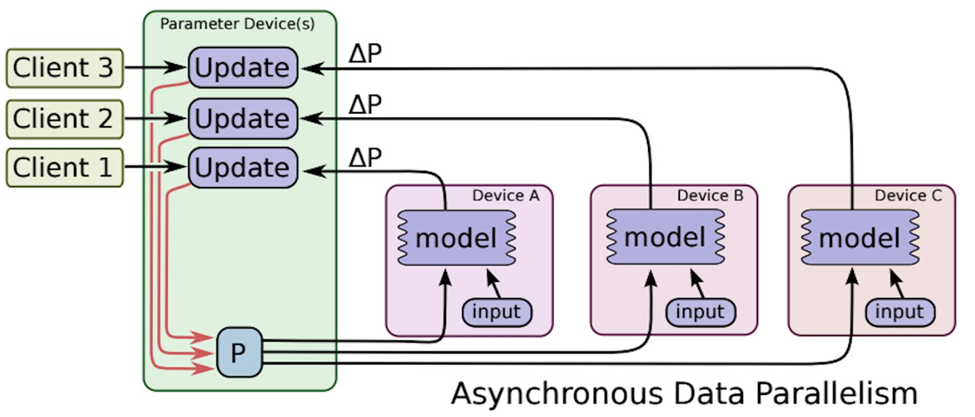
### 同步更新

这是最低效的方案， ps 会同时充当 reducer 的角色，等待所有节点都发来梯度和参数更新请求后，ps 会对梯度取平均(reduce mean)，并用平均过后的梯度更新一次参数。各个节点在从 ps 读取最新参数的过程中，以及等待 ps 更新参数的过程中，都是处于空闲状态。



### 异步更新

谷歌是最早使用ps和异步更新方案进行深度网络训练的。异步更新是ps模式的标配，其收敛性已经有证明。与同步更新不同，异步更新中ps在收到节点的梯度以及更新请求的时候，会立即对参数发起更新，而不等待其他节点。在完成梯度的计算后，节点会立刻从节点上读取参数，进行下一步的迭代。



异步更新将各个节点和ps的通信在时间上分散开，使得数据传输的等待时间减少。同时各个节点也不需要和其他节点同步，减少了阻塞的时间，特别是异构（多GPU为不同型号的设备）的节点产生的阻塞。这增加了模型训练的吞吐量。但是异步更新的方案会引入两个不稳定性来源：

* 参数和更新用的梯度并不来自同一个迭代。用来更新的梯度可能是几步更新前的参数算出来的。
* 参数的读取并没有加锁。这导致节点可能会读到更新一半的参数。

上述两个不稳定性来源要求模型采用更小的学习率。而小学习率加上上述的不稳定性会带来收敛速度的显著降低，同时训练发散的风险也增大了，这两者抵消了异步训练带来的吞吐量的提高。实际使用中经常会看到loss有时候会突然变得很高。所以异步更新目前已经不是主流优化方向了。

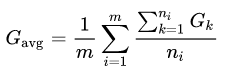
### Allreduce模式

介于异步更新可能导致的问题，提出了allreduce的更新模式。参数的变化来自于使用各个节点算出的梯度对其进行的更新。如果在初始化的时候便同步了所有节点拿到的参数拷贝，并在参数更新之前对梯度进行同步，这样在任何时刻各个节点的参数拷贝都是一致的。所以我们可以以梯度为同步基础，间接实现参数的同步。而梯度的同步则依赖于allreduce 操作(P2P 的通信)的高效实现。在Allreduce模式中，所有节点同时充当ps和worker的角色。

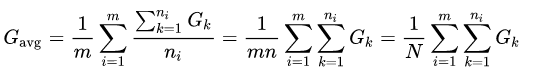
### 梯度平均

在多GPU训练中通常采用对梯度进行多级平均的方法：先对每个节点的 mini-batch 产生的梯度算一个平均，再对主机的各个节点算一次平均，最后再对多个节点算一个平均。下面针对多卡训练的情景进行分析。

假设第个节点的 mini-batch 的样本个数为，每个样本产生的梯度为，节点个数为，则多卡算得的平均梯度为：



假如各个节点的样本个数都相同则：



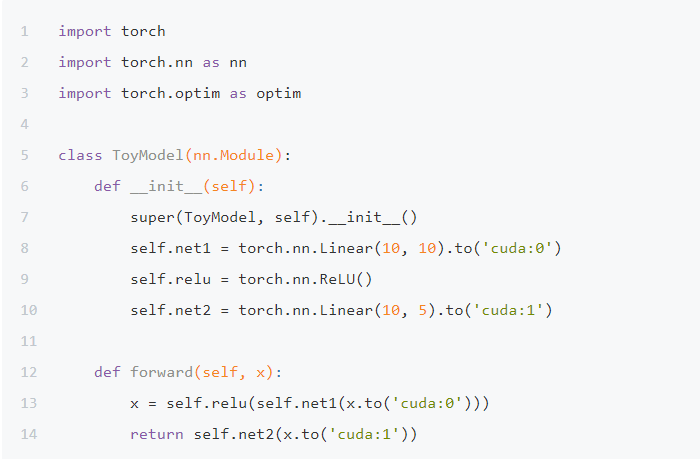
此时多卡/单卡对各个样本的梯度的权重是一致的，其中为一次更新中用到的总样本。而当各个节点的样本个数不等时，较大的节点中，每个样本的权重会低于，其中为所有的平均。在训练过程中这种影响通常忽略不计。

## 模型并行

虽然数据并行能极大的提高时间效率，但是不适用于某些模型太大而无法被单个GPU容纳的用例，因为每个GPU都需要加载全量的模型。

模型并行是将单个的模型拆分成k份，分发到k个节点上，每个节点只负责计算中的一部分。模型并行化的思想是将模型的不同子网络放置在不同的设备上，并相应地实现传播方法，以在节点之间传递中间输出。由于只需模型的一部分就能能在任何独立设备上运行，因此一组设备可以共同为更大的模型服务。

让我们从包含两个线性层的玩具模型开始。要在两个GPU上运行此模型，只需将每个线性层放在不同的GPU上，然后将输入和中间输出传递到匹配的层节点。



第二层分发到节点1上计算

第一层分发到节点0上计算

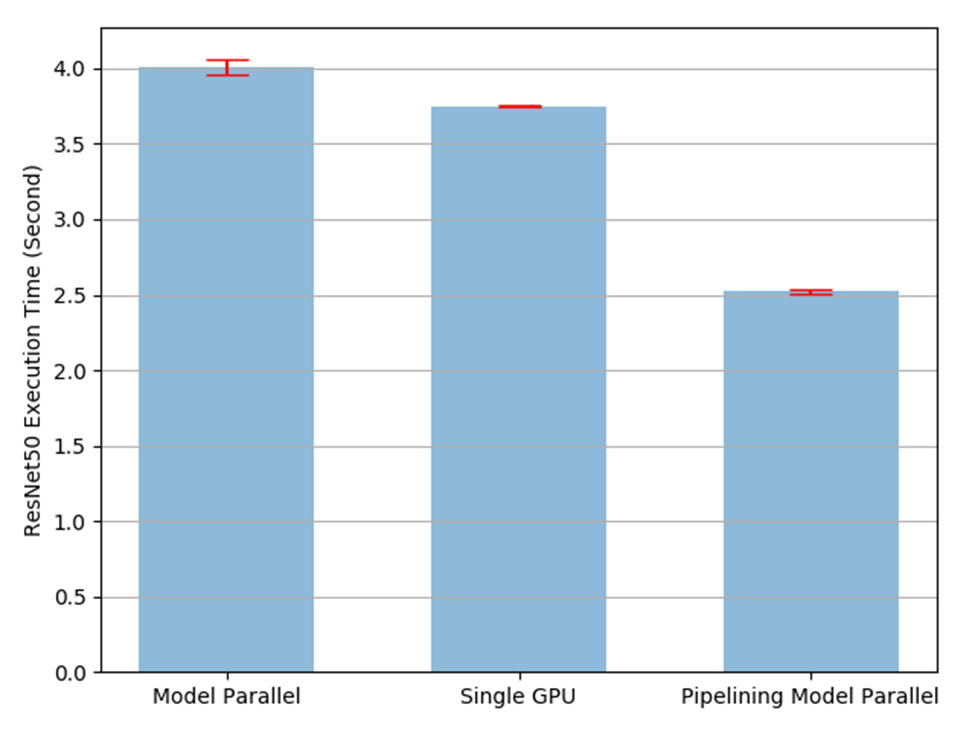
上面的玩具模型看起来非常类似于在单个GPU上实现它的方式。那是模型中唯一需要更改的地方。backward()和torch.optim将自动处理梯度，就像模型在一个GPU上一样。你只需确保调用损失函数时标签与输出在同一设备上。



## 比较

|  |  |
| --- | --- |
| 数据并行 | 模型并行 |
| 训练时间效率提高 | 能运行更大的模型 |
| 每个节点的数据不同 | 每个节点的数据相同 |
| 每个都有相同的模型 | 每个节点负责模型的一部分 |
| 模型的梯度需要通信交换 | 模型的激活函数需要通信交换 |

下图为Pytorch Resnet50的时间效率对比，左边为多卡模型并行，中间为单卡，右边为数据并行。



# 修订记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **变更时间** | **版本** | **变更人** | **审批人** | **变更说明** |
| 1 | 2020-2-19 | V0.1.0 | 万骏泓 |  | 新建 |
|  |  |  |  |  |  |