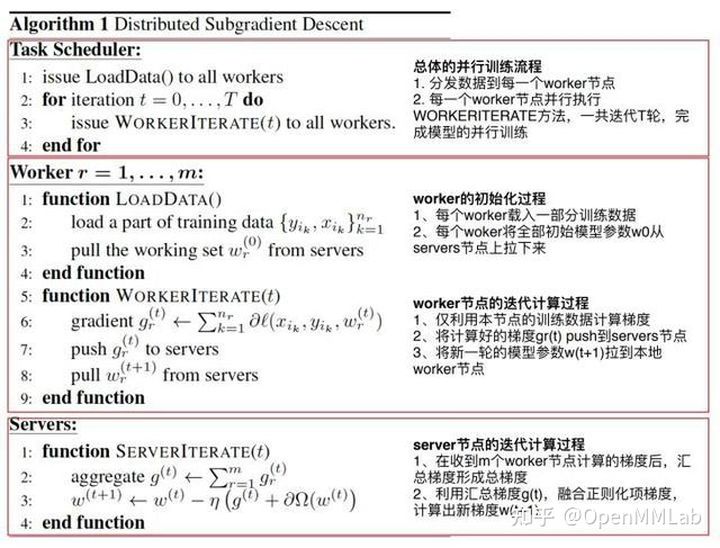
model **=** nn**.**DataParallel(model)

原理：



Parameter Server (PS)的并行梯度下降流程分为四个部分：

1. Task Scheduler: 负责加载数据并分发数据至每个worker节点， 并执行多轮迭代;

2. 在每轮迭代中，worker 负责：

初始化：载入数据并将全部模型参数从server 节点拉下来;

梯度计算：利用该节点的数据计算梯度并将梯度更新到server节点;

3.Server 负责：

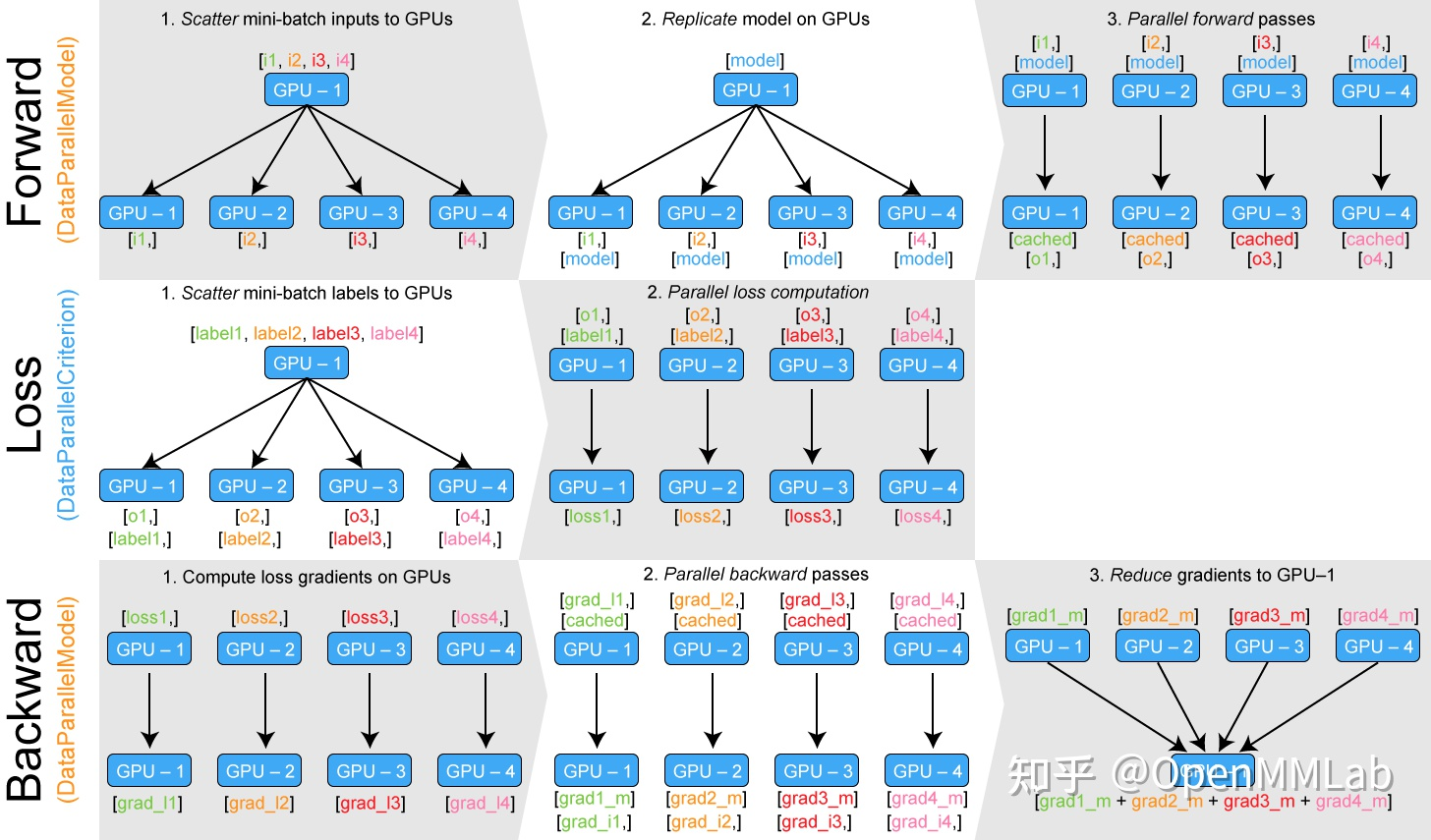
汇总梯度

更新参数

最后用一张图看DP Module究竟怎么执行的：

前向传播的时候我们会先用Scatter函数将数据从device[0]分配并复制到不同的卡，之后用Replicate函数将模型从device[0]复制到不同的卡，之后各个卡都有了同样的模型和不同的数据，分别调用forward计算损失和梯度。

反向传播的时候，我们会将梯度收集到device[0]然后在device[0]更新参数。



DDP （Distributed Data Parallel）

原理：

区别

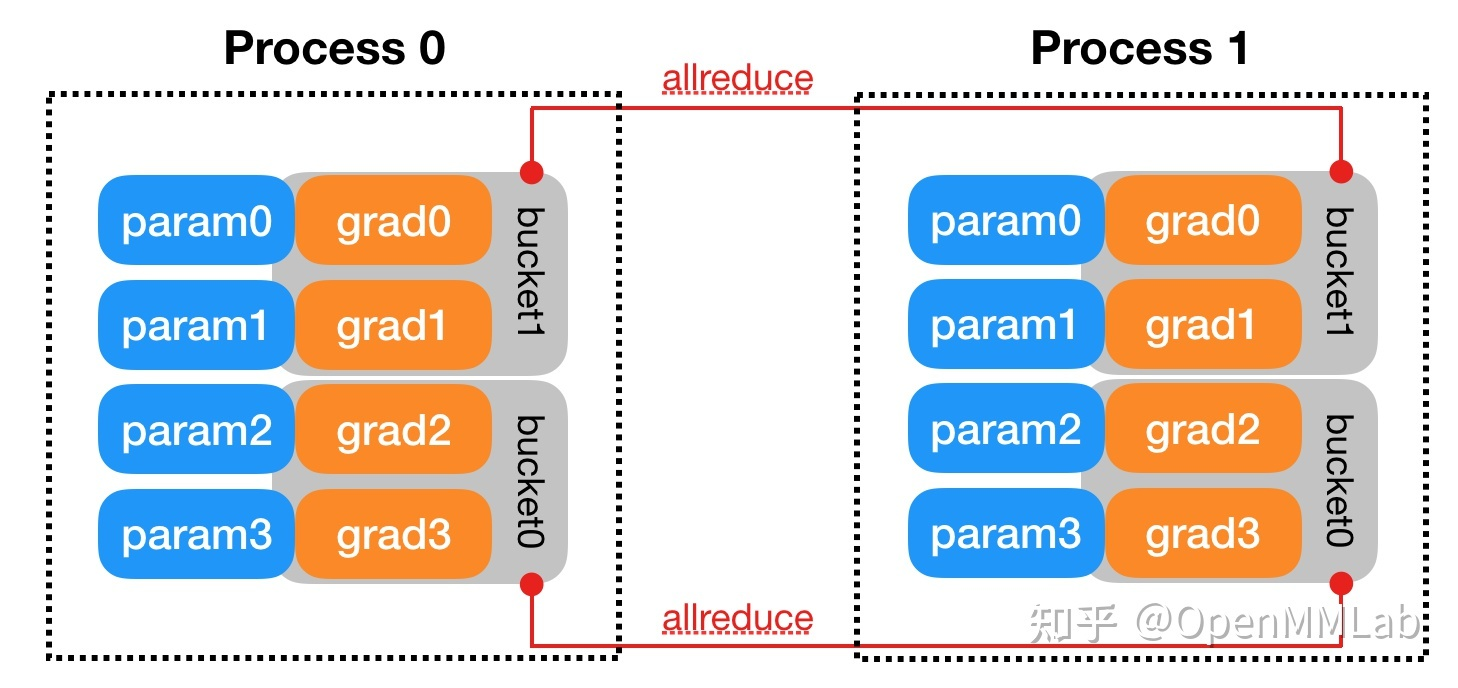
* + 多进程  
    和 DP 不同， DDP 采用多进程，最推荐的做法是每张卡一个进程从而避免上一节所说单进程带来的影响。前文也提到了 DP 和 DDP 共用一个 parallel\_apply 函数，所以 DDP 同样支持单进程多线程多卡操作，自然也支持多进程多线程，不过需要注意一下 world\_size。
  + 通信效率  
    DP 的通信成本随着 GPU 数量线性增长，而 DDP 支持 Ring AllReduce，其通信成本是恒定的，与 GPU 数量无关。
  + 同步参数  
    DP 通过收集梯度到 device[0]，在device[0] 更新参数，然后其他设备复制 device[0] 的参数实现各个模型同步；  
    DDP 通过保证初始状态相同并且改变量也相同（指同步梯度） ，保证模型同步。

**Ring AllReduce**

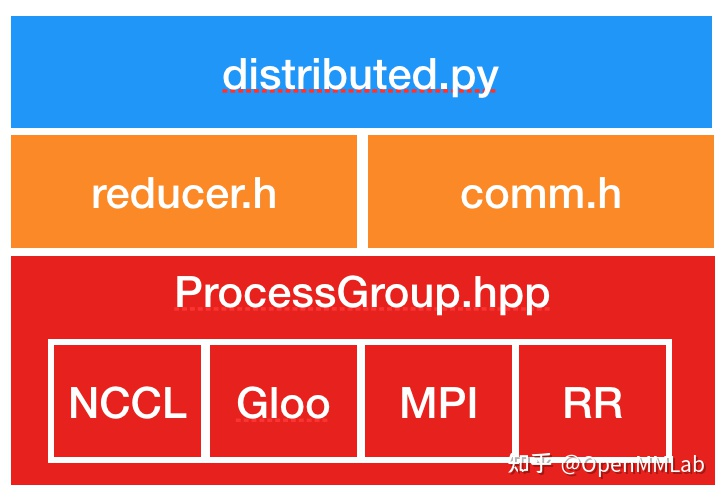
**DDP:**

DDP 也是数据并行，所以每张卡都有模型和输入。我们以多进程多线程为例，每起一个进程，该进程的 device[0] 都会从本地复制模型，如果该进程仍有多线程，就像 DP，模型会从 device[0] 复制到其他设备。

DDP 通过 Reducer 来管理梯度同步。为了提高通讯效率， Reducer 会将梯度归到不同的桶里（按照模型参数的 reverse order， 因为反向传播需要符合这样的顺序），一次归约一个桶。其中桶的大小为参数 bucket\_cap\_mb 默认为 25，可根据需要调整。下图即为一个例子。

DDP 通过在构建时注册 autograd hook 进行梯度同步。反向传播时，当一个梯度计算好后，相应的 hook 会告诉 DDP 可以用来归约。当一个桶里的梯度都可以了，Reducer 就会启动异步 allreduce 去计算所有进程的平均值。allreduce 异步启动使得 DDP 可以边计算边通信，提高效率。当所有桶都可以了，Reducer 会等所有 allreduce 完成，然后将得到的梯度写到 param.grad。

2.3实现

结构：

伪代码(Algorithm 1):

1. ***constructor*** steps:

a. broading casting model states

b. installing autograd hooks

2. ***forward***function is a simple wrapper of the local model’s forward. It traverses the autograd graph

to mark unused parameters accordingly.

3. The ***autograd\_hook*** takes the internal parameter tensor and its belonging bucket. It writes the

local gradient to the correct offset in the bucket and then launches the asynchronous AllReduce

operation.

