Προμηθεύς

【转载】 Ring Allreduce (深度神经网络的分布式计算范式 ---------- 环形全局规约)

作者：初七123

链接：https://www.jianshu.com/p/8c0e7edbefb9

来源：简书

(深度神经网络的分布式计算范式 ---------- 环形全局规约) （副标题 自起）

The Communication Problem

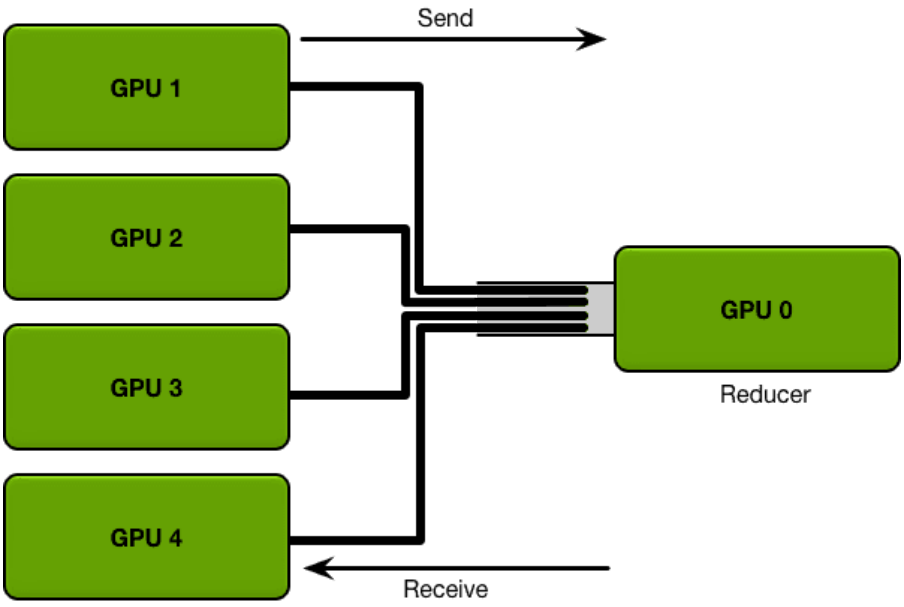
当将神经网络的训练并行化到许多GPU上时，你必须选择如何将不同的操作分配到你可用的不同GPU上。在这里，我们关注一种称为数据并行随机梯度下降(SGD)的技术。与标准SGD一样，梯度下降是通过数据子集(小批次)完成的，需要多次迭代才能在整个数据集上进行。然而，在数据并行训练中，每个GPU都有整个神经网络模型的完整副本，对于每次迭代，只分配了小批次中样本的子集。对于每次迭代，每个GPU在其数据上运行网络的前向传播，随后进行误差反向传播，以计算损耗相对于网络参数的梯度。最后，GPU相互通信以平均由不同GPU计算的梯度，将平均梯度应用于权重以获得新权重。GPU都在锁定步骤的迭代中前进，一旦GPU完成了迭代，它必须等待所有其他GPU完成它们的迭代，这样权重才能被正确更新。这相当于在单个GPU上执行SGD，但是我们通过在多个GPU之间分发数据并行执行计算来获得加速。

当你只有两个GPU和以兆字节数据衡量的参数时，这些GPU的通信方式可能并不重要。然而，当你的模型有数十亿个参数时，梯度可能需要几十亿字节的空间(因为每个参数都有一个

例如，考虑最直接的通信机制。每一个GPU都计算其子集的小批次上的梯度。然后，每个GPU将其梯度发送到单个GPU，该GPU取所有梯度的平均值，并将平均值发送回所有其他GPU。

在直接从单个GPU发送和接收数据的机制中，单个GPU必须从所有GPU接收所有参数，并将所有参数发送到所有GPU。系统中的gpu越多，通信成本就越大。

让我们评估一下这种通信策略如何在真实模型上运行，例如以**百度深度语音2** 为模型的语音识别网络，具有三亿个可训练参数。 每个参数四个字节的三亿个参数大约是1.2千兆字节的数据。 假设您系统上的网络硬件可以支持每秒1千兆字节的带宽; 在这种情况下，如上所述将系统并行化到两个GPU上将使每次迭代减慢1.2秒。 将您的训练并行化到10个GPU将使每次迭代减慢10.8秒; 随着GPU数量的增长，每次迭代所需的时间呈线性增长。 即使每次迭代花费几秒钟，通信成本的这种线性增长也会使得进一步的并行化变得不切实际并且会降低训练效率。



需要发送的数据越多，发送时间就越长;每个通信通道都有一个最大的吞吐量(带宽)。例如，一个好的internet连接可以提供**每秒15兆字节**的带宽，而千兆以太网连接可以提供**每秒125兆字节**的带宽。HPC集群上的专用网络硬件(如**Infiniband**)可以在节点之间提供每秒数**gb的带宽**。

25	26	27	28	29	30	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

统计

随笔 - 1262
文章 - 0
评论 - 329
阅读 - 214万

搜索

找找看
谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

杂谈(578)
Linux(121)
强化学习(98)
Python(84)
Tensorflow(60)
机器学习(53)
MindSpore（深度学习计算框架）(37)
PyTorch(32)
国家科技项目(32)
遗传算法(23)
更多

积分与排名

积分 - 1083068
排名 - 272

随笔档案

2023年7月(13)
2023年6月(23)
2023年5月(19)
2023年4月(25)
2023年3月(17)
2023年2月(19)
2023年1月(48)
2022年12月(17)
2022年11月(31)
2022年10月(23)
2022年9月(13)
2022年8月(15)
2022年7月(22)
2022年6月(31)
2022年5月(29)
更多

阅读排行榜

1. c++ 中 char 与 string 之间的相互转换问题(185905)

另一种选择是**放弃训练算法的同步性**，并通过梯度下降的迭代消除所有GPU在锁定步骤中前进的限制。然而，虽然这可以使模型更容易并行化，但是**消除这种约束的算法**(异步SGD的变体)可能很难调试，对于某些模型来说，可能会收敛到子结果，所以我们不考虑这些问题。

相反，我们可以通过使用**高性能计算领域**的**分布式缩减算法**并利用**带宽优化环**来解决通信问题。

The Ring Allreduce

上述 简单 通信策略 的主要问题是，通信成本随系统中gpu的数量线性增长。相比之下，环allreduce算法的通信成本是恒定的，**与系统中gpu的数量无关**，完全由系统中gpu之间最慢的连接决定;事实上，如果您只考虑带宽作为通信成本的一个因素(**并忽略延迟**)，那么**环allreduce**是一种最优通信算法（当您的模型很大，并且您需要发送大量数据的次数很少时，这是一个很好的通信成本估算。）。

环中的gpu都被安排在一个逻辑环中。每个GPU应该有一个左邻和一个右邻;它只会向它的右邻居发送数据，并从它的左邻居接收数据。

g)（84079)
3. python 库 Numpy 中如何求取向量范数 np.linalg.norm(求范数)（向量的第二范数为传统意义上的向量长度），（如何求取向量的单位向量）(67797)
4. 安装 aconda 后Linux的终端界面前部出现（base）字样(46977)
5. python 编程中的一个关于图片的库 imageio (读取照片RGB内容，转换照片格式)(40011)

评论排行榜

1. 浪潮计算平台之AI方向——AI_Station开发环境的使用总结(26)

推荐排行榜

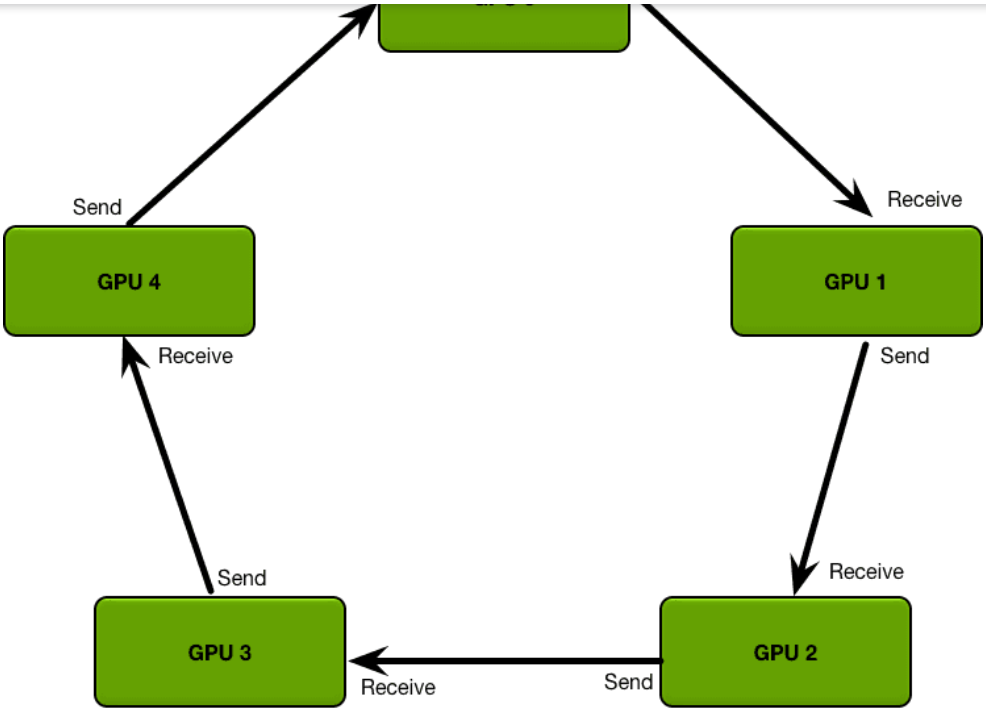
1. 安装 aconda 后Linux的终端界面前部出现（base）字样(8)
2. c++ 中 char 与 string 之间的相互转换问题(5)
3. 动态规划中 策略迭代 和 值迭代 的一个小例子(4)
4. python 库 Numpy 中如何求取向量范数 np.linalg.norm(求范数)（向量的第二范数为传统意义上的向量长度），（如何求取向量的单位向量）(3)
5. Centos7 服务 service 设置命令 systemctl 用法 （替代service 和 chkconfig）(3)

最新评论

1. Re:从图像中检测和识别表格，北航＆微软提出新型数据集 TableBank

@Death_Knight 源地址好像下载不了了，大佬还有这个数据集吗...

--虚先生

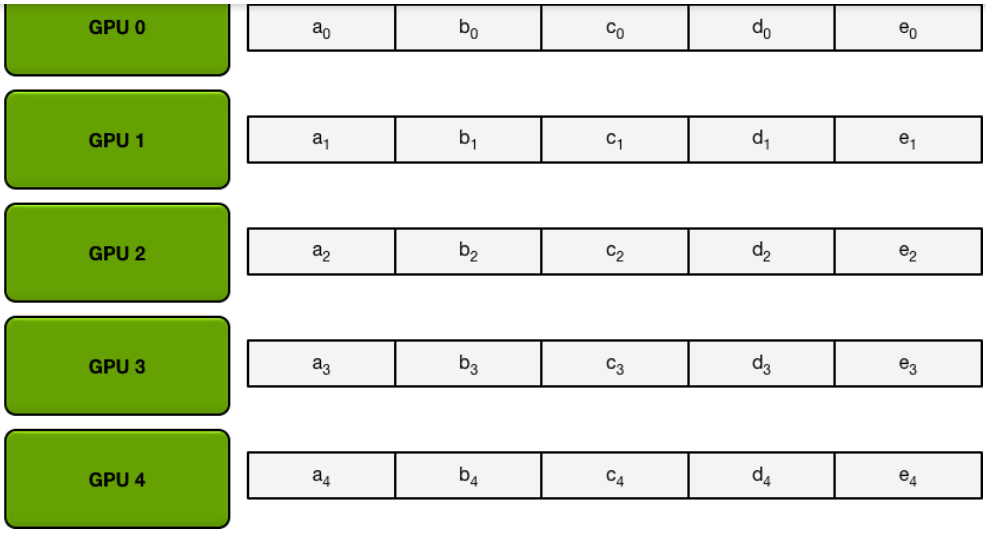


该算法分两个步骤进行:首先是**scatter-reduce**，然后是**all-gather**。在**scatter-reduce**步骤中，GPU将交换数据，使每个GPU可得到最终结果的一个块。在**all-gather**步骤中，gpu将交换这些块，以便所有gpu得到完整的最终结果。

The Scatter-Reduce

为简单起见，让我们假设目标是对一个浮点数的大数组求和; 系统中有N个GPU，**每个GPU都有一个相同大小的数组**，并且在all-reduce阶段的最后，每个GPU都应该有一个相同大小的数组，其中包含原始数组中数字的总和。

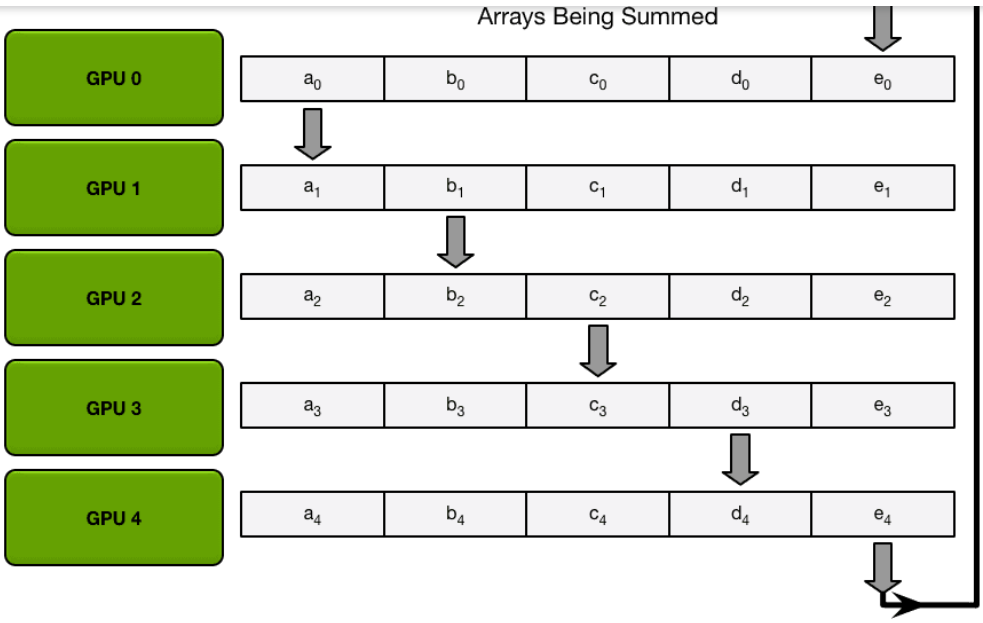
首先，gpu将数组划分为N个更小的块(其中N是环中的gpu数)。



接下来，GPU将进行**N-1**次 **Scatter-Reduce** 迭代；在每次迭代中，GPU将向其右邻居发送一个块，并从其左邻居接收一个块并累积到该块中。每个GPU发送和接收的块在每次迭代中都是不同的；第n个GPU从发送块N和接收块N - 1开始，然后从那里向后进行，每次迭代都发送它在前一次迭代中接收到的块。

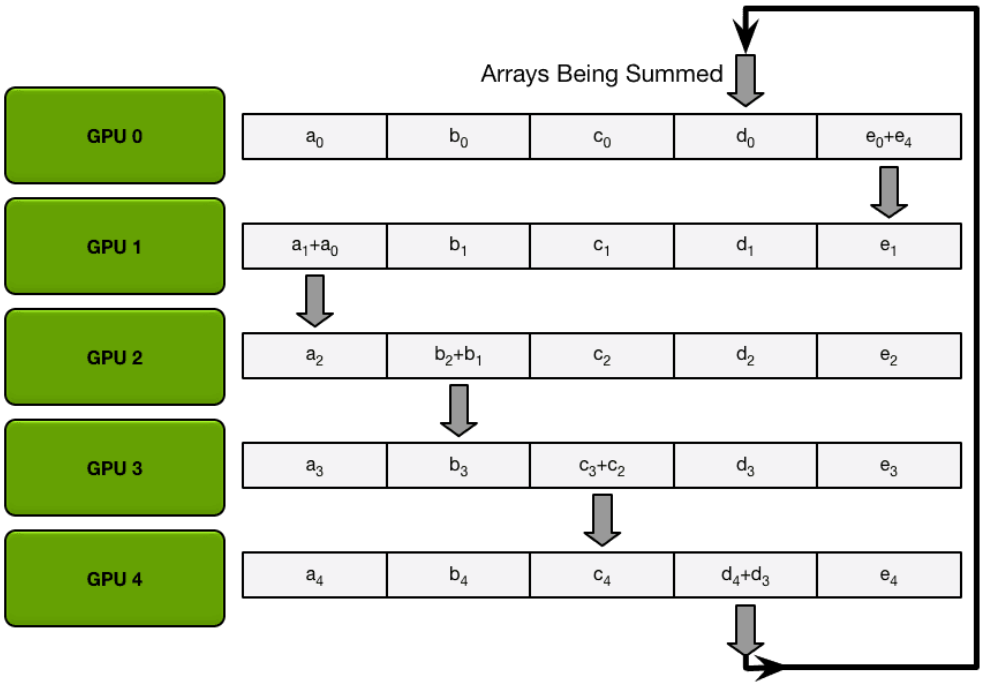
例如，在第一次迭代中，上图中的五个GPU将发送和接收以下区块：

GPU	Send	Receive
0	Chunk 0	Chunk 4
1	Chunk 1	Chunk 0
2	Chunk 2	Chunk 1
3	Chunk 3	Chunk 2
4	Chunk 4	Chunk 3



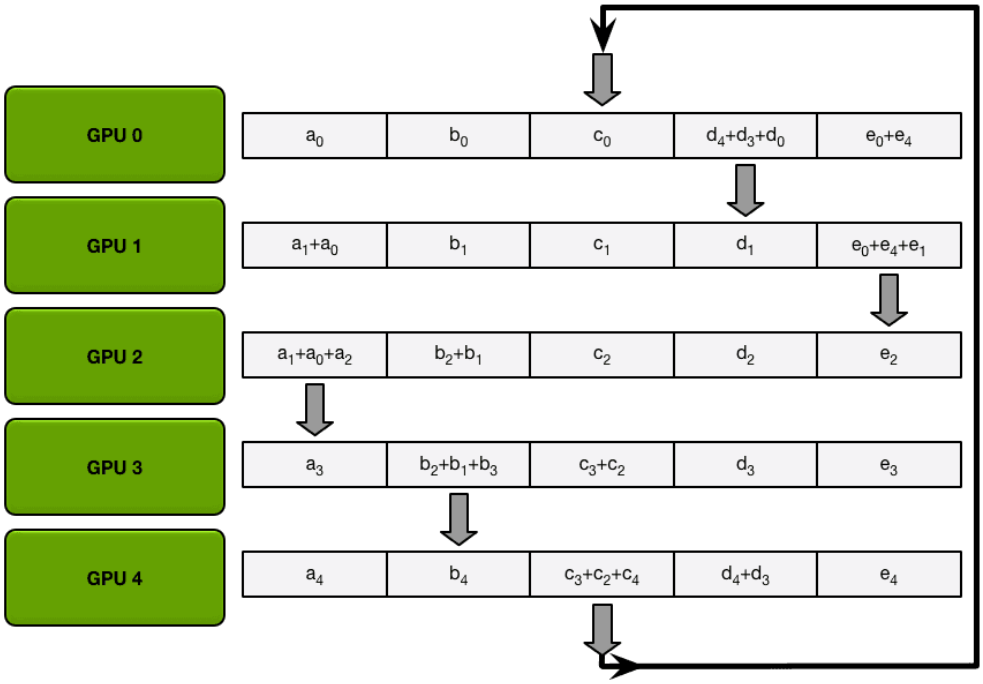
Data transfers in the first iteration of scatter-reduce

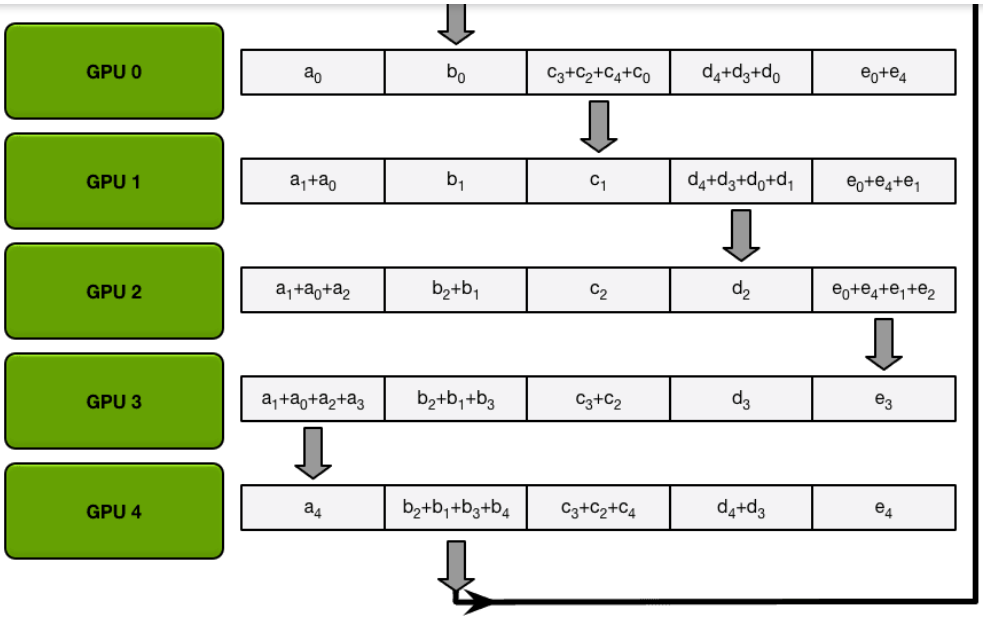
在第一次发送和接收完成之后，每个GPU将拥有一个块，该块由两个不同GPU上相同块的和组成。例如，第二个GPU上的第一个块将是该块中来自第二个GPU和第一个GPU的值的和。

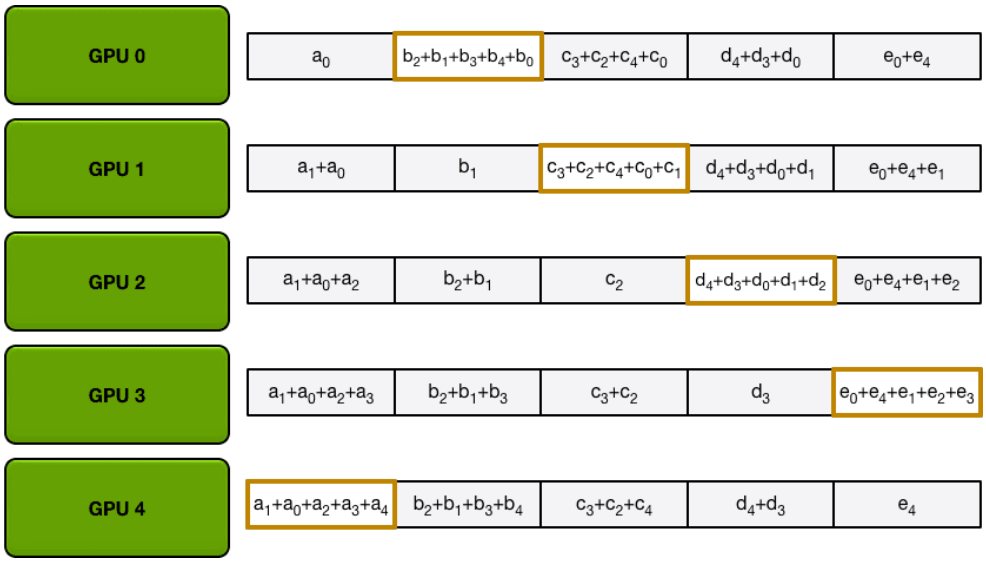


Itermediate sums after the first iteration of scatter-reduce is complete

在下一次迭代中，该过程继续进行，到最后，每个GPU将有一个块，该块包含所有GPU中该块中所有值的总和。下图展示了所有数据传输和中间结果，从第一次迭代开始，一直持续到**Scatter-Reduce**完成。







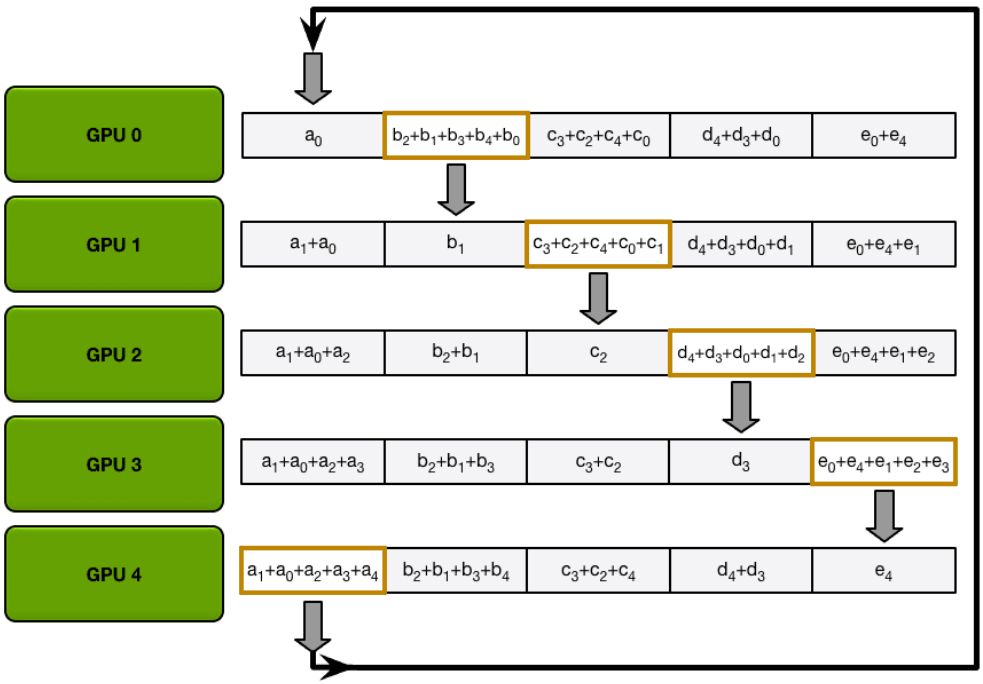
The Allgather

在**scatter-reduce**步骤完成之后，每个GPU都有一个值数组，其中一些值(每个GPU一个块)是最终的值，其中包括来自所有GPU的贡献。为了完成**all-reduce**, gpu必须交换这些块，以便所有gpu都具有所有必需的值。

环的收集过程与**scatter-reduce**是相同的(发送和接收的N-1次迭代)，只是gpu接收的值没有累加，而是简单地覆盖块。第n个GPU首先发送第n+1个块并接收第n个块，然后在以后的迭代中总是发送它刚刚接收到的块。

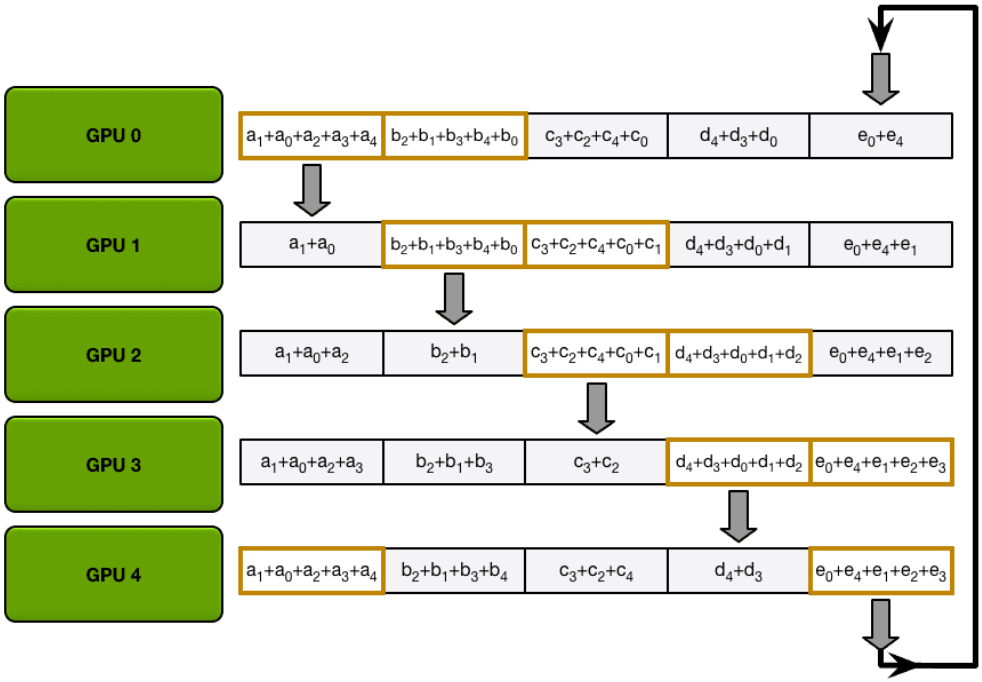
例如，在我们的5 - gpu设置的第一次迭代中，gpu将发送和接收以下块

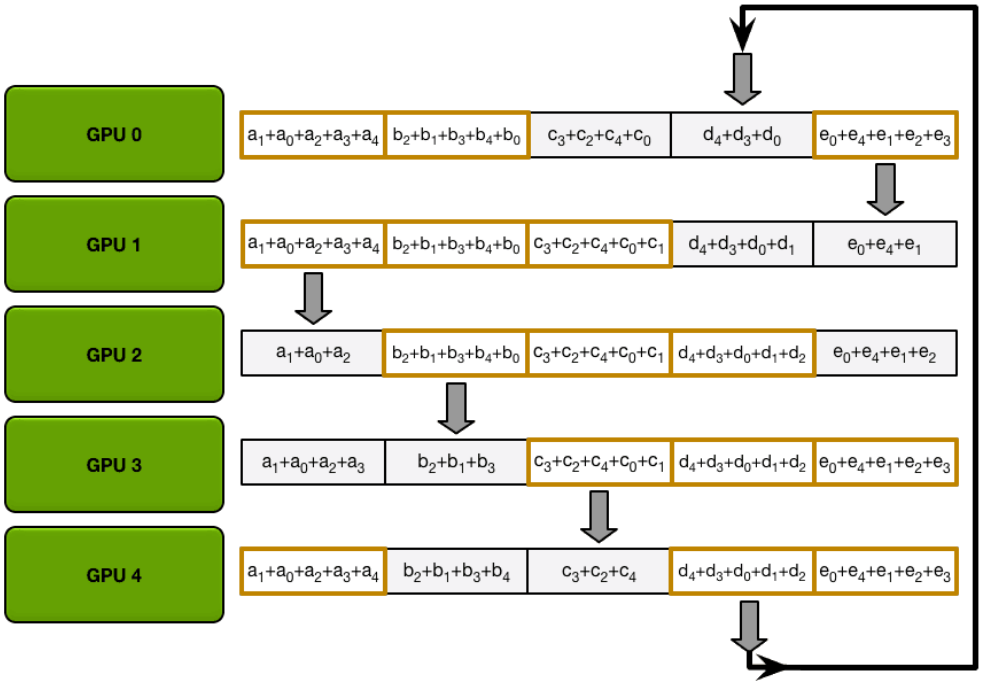
GPU	Send	Receive
0	Chunk 1	Chunk 0
1	Chunk 2	Chunk 1
2	Chunk 3	Chunk 2
3	Chunk 4	Chunk 3
4	Chunk 0	Chunk 4

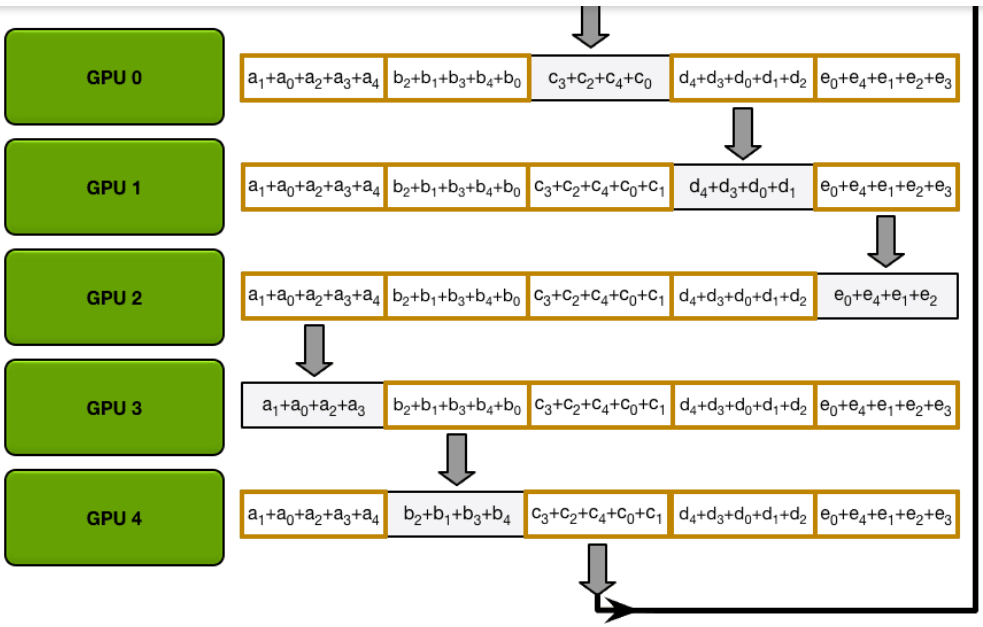


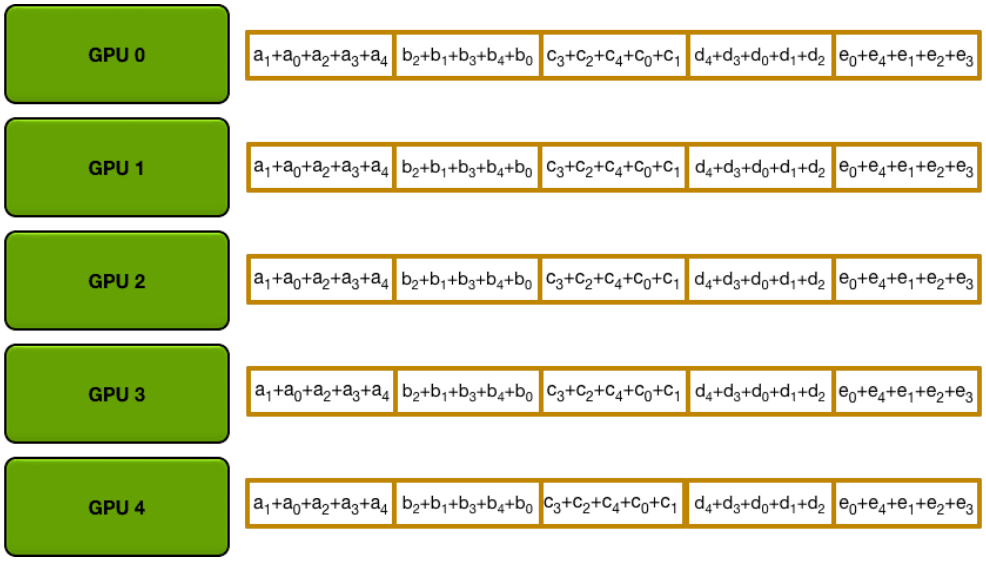
第一次迭代完成后，每个GPU将拥有最终数组的两个块。

在下一个迭代中，该过程将继续，到最后，每个GPU将拥有整个数组的完整累积值。下面的图像演示了所有数据传输和中间结果，从第一次迭代开始，一直到**all-gather**完成。






GPU 0
GPU 1
GPU 2
GPU 3
GPU 4
$a_1+a_0+a_2+a_3+a_4$ $b_2+b_1+b_3+b_4+b_0$ $c_3+c_2+c_4+c_0$ $d_4+d_3+d_0+d_1+d_2$ $e_0+e_4+e_1+e_2+e_3$
$a_1+a_0+a_2+a_3+a_4$ $b_2+b_1+b_3+b_4+b_0$ $c_3+c_2+c_4+c_0+c_1$ $d_4+d_3+d_0+d_1$ $e_0+e_4+e_1+e_2+e_3$
$a_1+a_0+a_2+a_3+a_4$ $b_2+b_1+b_3+b_4+b_0$ $c_3+c_2+c_4+c_0+c_1$ $d_4+d_3+d_0+d_1+d_2$ $e_0+e_4+e_1+e_2$
$a_1+a_0+a_2+a_3$ $b_2+b_1+b_3+b_4+b_0$ $c_3+c_2+c_4+c_0+c_1$ $d_4+d_3+d_0+d_1+d_2$ $e_0+e_4+e_1+e_2+e_3$
$a_1+a_0+a_2+a_3+a_4$ $b_2+b_1+b_3+b_4$ $c_3+c_2+c_4+c_0+c_1$ $d_4+d_3+d_0+d_1+d_2$ $e_0+e_4+e_1+e_2+e_3$


GPU 0
GPU 1
GPU 2
GPU 3
GPU 4
$a_1+a_0+a_2+a_3+a_4$ $b_2+b_1+b_3+b_4+b_0$ $c_3+c_2+c_4+c_0+c_1$ $d_4+d_3+d_0+d_1+d_2$ $e_0+e_4+e_1+e_2+e_3$

回想一下，对于介绍中描述的简单通信算法，通信成本随着GPU的数量线性增长。 **all-reduce**运行良好的主要原因是不再是这种情况。

在我们描述的系统中，N个GPU中的每一个都将发送和接收**N-1**次**scatter-reduce**，**N-1**次**all-gather**。每次，GPU都会发送**K / N**值，其中K是数组中不同GPU上相加的值总数（K为每个GPU上所需存储数组的总大小）。因此，传输到每个GPU和从每个GPU传输的数据总量为

$$\text{Data Transferred} = 2(N-1)\frac{K}{N}$$

重要的是，这与GPU的数量无关。

由于所有传输都是在离散迭代中同步进行的，因此所有传输的速度受到环中相邻GPU之间最慢(最低带宽)连接的限制。给定每个GPU的邻居的正确选择，该算法是带宽最优的，并且是执行全面操作的最快算法**(**假设延迟成本与带宽相比可以忽略不计**)**。一般来说，如果一个节点上的所有GPU在环中彼此相邻，则该算法的功能最佳；这最小化了网络争用的量，否则这可能会显著降低GPU-GPU连接的有效带宽。

Applying the Allreduce to Deep Learning

Ring all-reduce 是高性能计算领域中著名的算法，但在深度学习中很少使用。在我们的实验室中，我们已经成功地将这个工具作为所有数据并行训练的基础，使我们能够有效地将训练扩展到几十个gpu。

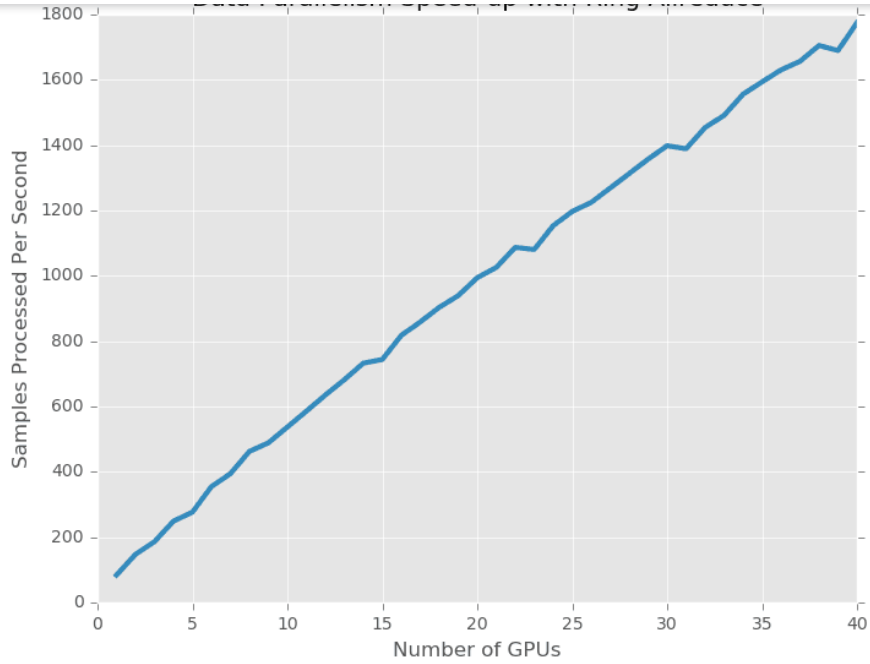
播来计算误差，然后进行反向传播来计算神经网络的每个参数的梯度。反向传播计算梯度，从输出层开始，向输入层移动，这意味着输出层参数的梯度在早期层的梯度计算出之前很明显是可用的。因为全部运算可以一次对网络的一部分参数进行运算，所以我们可以在其他梯度仍在计算的时候开始对输出层参数进行全部运算。这样做将通信与反向传播步骤中的其余计算重叠，从而减少了每个GPU等待通信完成的总时间。

例如，考虑一个类似于2的语言模型，但有大约3亿个可学习的参数（因此总梯度大小为1.2千兆字节）。 使用**all-reduce**，每个GPU必须发送和接收大约2.4千兆字节的数据。 使用支持CUDA的MPI实现（例如OpenMPI），我们可以使用**GPUDirect RDMA**在GPU之间传输数据，带宽大约为每秒10千兆字节; 但是，我们集群中节点之间的连接速度较慢，**Infiniband**提供的带宽大约为每秒6千兆字节。 由于限制因素是Infiniband连接，因此单次迭代需要大约

$$\frac{2.4 \text{ gigabytes}}{6.0 \text{ gigabytes per second}} \approx 400 \text{ milliseconds per iteration}$$

由于更深层次的网络首先有可用的梯度，我们可以在完成整个反向传播传递之前开始进行数据传输，因此真正的开销可能小于400毫秒;根据所优化的神经网络的性质，通信和计算之间的重叠可能有所不同。

我们实现了上述语言模型，并测试了每次迭代所花费的时间，因为我们从单个GPU（没有通信开销）扩展到40个GPU。 这40个GPU排列成**5个节点**，**每个节点有8个GPU**，由Infiniband连接。 我们运行语言模型300次迭代，批量大小为32，并计算每秒处理的样本数。



正如您所看到的，整个系统的吞吐量随着GPU的数量线性扩展；超过一定的节点后，添加更多的GPU不会导致每次迭代的显著减速。在40个GPU上运行模型每次迭代大约需要650 - 700毫秒，而在单个GPU上大约需要370毫秒。根据我们的估计，通信将花费400毫秒，通过将反向传播与数据传输重叠，我们在每次迭代中节省了额外的70 - 120毫秒。

转者注：

实现本文深度神经网络的ring all-reuce模式多数成型框架一般需采用NVIDIA公司的nccl框架。

标签: 分布式系统 , Tensorflow , PyTorch , MPI 高性能并行计算 , 神经网络

Death_Knight
粉丝 - 145 关注 - 18

1 0

+加关注

« 上一篇： 【转载】 tensorflow batch_normalization的正确使用姿势
» 下一篇： 【转载】 Tensorflow Guide: Batch Normalization （tensorflow中的Batch Normalization）

posted on 2020-03-09 09:49 Death_Knight 阅读(1819) 评论(1) 编辑 收藏 举报