(19)国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 115794387 A
(43)申请公布日 2023.03.14

(21)申请号 202211431240.8

(22)申请日 2022.11.14

(71)申请人 苏州国科综合数据中心有限公司
地址 215000 江苏省苏州市工业园区星湖街328号创意产业园A2幢

(72)发明人 蒋鹏飞 单晓冬 徐恩格 王小龙 鲍复劼

(74)专利代理机构 北京同恒源知识产权代理有限公司 11275
专利代理师 杨慧红

(51)Int.Cl.
G06F 9/50(2006.01)
G06N 3/08(2023.01)

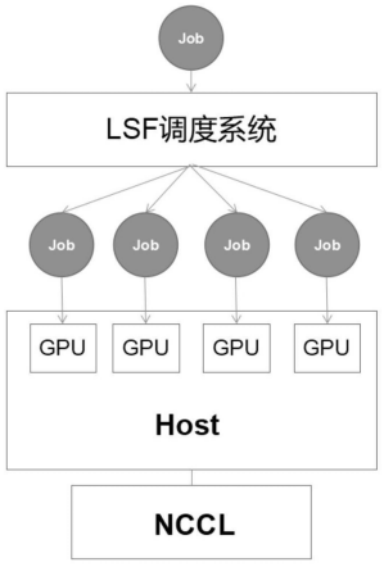
权利要求书1页 说明书4页 附图1页

(54)发明名称

基于LSF的单主机多GPU分布式pytorch并行计算方法

(57)摘要

本发明涉及一种基于LSF的单主机多GPU分布式pytorch并行计算方法,属于计算机领域。该方法包括两个部分:第一部分:资源申请及调度;第二部分:使用资源进行深度学习模型的训练。使用一个进程来计算模型参数,然后在每个批处理期间将分发到每个GPU,每个GPU计算各自的梯度,汇总到GPU0中进行求平均,由GPU0进行反向传播更新参数,再把模型的参数由GPU0传播给其他的GPU。GPU利用率通常很低。nn.DataParallel要求所有的GPU都在同一个节点上,而且不能使用Apex进行混合精度训练。相较于现有的dataparrell方式,速度更快,效率高,GPU占用更高。



CN 115794387 A

1.基于LSF的单主机多GPU分布式pytorch并行计算方法,其特征在于:该方法包括两个部分:

第一部分:资源申请及调度;

第二部分:使用资源进行深度学习模型的训练。

2.根据权利要求1所述的基于LSF的单主机多GPU分布式pytorch并行计算方法,其特征在于:所述第一部分在LSF集群下完成;

通过LSF的指令申请计算资源,包括:

要创建的作业总数,其值等于申请的GPU总数;

单台主机的GPU个数。

3.根据权利要求2所述的基于LSF的单主机多GPU分布式pytorch并行计算方法,其特征在于:所述第二部分在程序内部实现;

首先每个LSF作业独占一个进程和GPU,深度学习模型基于pytorch框架;

第一步,每个作业从环境中读取'LSF_PM_TASKID'作为每个任务的rank;

第二步,使用torch.distributed库初始化分布式进程组,参数包括rank,world_size,init_method,backend;rank用来指代每一个进程,world_size是进程总数,init_method用于表明从何处及如何发现其他进程,backend用于指明使用的后端通信方法,该发明中使用nccl作为通信后端;NCCL是NVIDIA公司为GPU并行计算开发的通信后端;

第三步,读取训练用数据集;数据集采用pytorch中的torch.utils.data.distributed.Distributed Sampler方法进行切分,通过设置num_replicas为world_size及rank为当前进程的rank,使每个进程获取相应的数据切片;如果使用Distributed Sampler进行数据切片,那么每个进程上训练的批的尺寸需除以进程总数;使用torch.utils.data.DataLoader进一步读取每个数据切片上数据;其中num_workers设为大于1的数用于开启子进程加快数据读取速度,pin_memory设为True用于将数据直接读到进程所独占的GPU上,减少数据在传输时的耗时;

第四步,导入相关包,包括argparse、distributed、Distributed Data Parallel,导入成功后加一个参数,local_rank;告知当前的程序跑在那一块GPU上;最后是指定通信方式,选nccl;

第五步,包装Dataloader;这里需要的是将sampler改为Distributed Sampler,然后赋给Data Loader里面的sampler;每个GPU或者每个进程都会从DataLoader里面取数据,指定Distributed Sampler能够让每个GPU取到不重叠的数据;

第六步,使用Distributed Data Parallel包装模型;

第七步,将输入数据放到指定GPU。

基于LSF的单主机多GPU分布式pytorch并行计算方法

技术领域

[0001] 本发明属于计算机领域,涉及基于LSF的单主机多GPU分布式pytorch并行计算方法。

背景技术

[0002] 近些年来,深度学习技术在图像和自然语言处理等方向发展迅速。为了使模型具有更高的精度和更强的泛化能力,在设计时模型结构往往更深更复杂,训练用的数据也更庞大。其中模型迭代时的前向传播与反向传播步骤伴随着大量计算,是典型的计算密集型任务。尽管硬件上GPU(Graphics Processing Unit-图形处理器)可以提供更强的算力,模型本身可以通过算法进行优化,加快收敛速度,但单机能提供的资源依旧无法满足大规模的训练任务。分布式计算通过将训练任务切分并使用多节点并行执行可以有效缓解了这一问题。

[0003] PyTorch是一个开源的Python机器学习库,基于Torch,用于自然语言处理等应用程序。2017年1月,由Facebook人工智能研究院(FAIR)基于Torch推出了PyTorch。它是一个基于Python的可续计算包,提供两个高级功能:

[0004] 1、具有强大的GPU加速的张量计算(如NumPy)。

[0005] 2、包含自动求导系统的深度神经网络。

[0006] LSF(load sharing facility)是IBM旗下一个工业导向,商业级的软件。它强大的资源调度管理能力,使其能以更快的速度,更均衡的负载,更可靠的表现及更低的成本去分配多种IT资源执行分布式任务。对于深度学习训练任务,LSF能高效灵活地分配GPU资源,帮助创建,管理分布式计算环境,从而加快训练速度。但对于LSF环境下的作业,加速神经网络模型的训练最简单直接的办法是使用GPU,往往一块GPU的显存容量是有限的,如果遇到了比较大的模型或者是参数量巨大的情况下会直接爆显存,最简单粗暴的办法就是加GPU。这会有一个核心的问题,为了使用多块GPU进行训练,必须想一个办法在多个GPU上进行分发数据和模型,并且协调训练的过程,提升训练速度。

[0007] 目前PyTorch可以在单块GPU卡中进行计算,但无法通过多块GPU并行计算达到和单块GPU同等的倍数计算效率,使得多卡的使用性价比大大降低。

发明内容

[0008] 有鉴于此,本发明的目的在于提供一种基于LSF的单主机多GPU分布式pytorch并行计算方法,解决单任务无法在超算中心调用多块GPU计算资源的技术问题,使软件在多GPU平台上提高运算能力,且该项目在任意超算中心均可使用。

[0009] 为达到上述目的,本发明提供如下技术方案:

[0010] 基于LSF的单主机多GPU分布式pytorch并行计算方法,该方法包括两个部分:

[0011] 第一部分:资源申请及调度;

[0012] 第二部分:使用资源进行深度学习模型的训练。

[0013] 所述第一部分在LSF集群下完成；

[0014] 通过LSF的指令申请计算资源，包括：

[0015] 要创建的作业总数，其值等于申请的GPU总数；

[0016] 单台主机的GPU个数。

[0017] 可选的，所述第二部分在程序内部实现；

[0018] 首先每个LSF作业独占一个进程和GPU，深度学习模型基于pytorch框架；

[0019] 第一步，每个作业从环境中读取'LSF_PM_TASKID'作为每个任务的rank；

[0020] 第二步，使用torch.distributed库初始化分布式进程组，参数包括rank，world_size，init_method，backend；rank用来指代每一个进程，world_size是进程总数，init_method用于表明从何处及如何发现其他进程，backend用于指明使用的后端通信方法，该发明中使用nccl作为通信后端；NCCL是NVIDIA公司为GPU并行计算开发的通信后端；

[0021] 第三步，读取训练用数据集；数据集采用pytorch中的torch.utils.data.distributed.Distributed Sampler方法进行切分，通过设置num_replicas为world_size及rank为当前进程的rank，使每个进程获取相应的数据切片；如果使用Distributed Sampler进行数据切片，那么每个进程上训练的批的尺寸需除以进程总数；使用torch.utils.data.DataLoader进一步读取每个数据切片上数据；其中num_workers设为大于1的数用于开启子进程加快数据读取速度，pin_memory设为True用于将数据直接读到进程所独占的GPU上，减少数据在传输时的耗时；

[0022] 第四步，导入相关包，包括argparse、distributed、Distributed Data Parallel，导入成功后加一个参数，local_rank；告知当前的程序跑在那一块GPU上；最后是指定通信方式，选nccl；

[0023] 第五步，包装Dataloader；这里需要的是将sampler改为Distributed Sampler，然后赋给Data Loader里面的sampler；每个GPU或者每个进程都会从DataLoader里面取数据，指定Distributed Sampler能够让每个GPU取到不重叠的数据；

[0024] 第六步，使用Distributed Data Parallel包装模型；

[0025] 第七步，将输入数据放到指定GPU。

[0026] 本发明的有益效果在于：

[0027] DataParallel更易于使用(只需简单包装单GPU模型)因此成为主流的单节点多卡使用方式。model＝nn.DataParallel(model)它使用一个进程来计算模型参数，然后在每个批处理期间将分发到每个GPU，每个GPU计算各自的梯度，之后汇总到GPU0中进行求平均，由GPU0进行反向传播更新参数，再把模型的参数由GPU0传播给其他的GPU。这种使用方式通信速度成为一个瓶颈，GPU利用率通常很低。nn.DataParallel要求所有的GPU都在同一个节点上(不支持分布式)，而且不能使用Apex进行混合精度训练。相较于现有的dataparrell方式，速度更快，效率高，GPU占用更高。

[0028] 本发明的其他优点、目标和特征在某种程度上将在随后的说明书中进行阐述，并且在某种程度上，基于对下文的考察研究对本领域技术人员而言将是显而易见的，或者可以从本发明的实践中得到教导。本发明的目标和其他优点可以通过下面的说明书来实现和获得。

附图说明

[0029] 为了使本发明的目的、技术方案和优点更加清楚，下面将结合附图对本发明作优选的详细描述，其中：

[0030] 图1为本发明原理图。

具体实施方式

[0031] 以下通过特定的具体实例说明本发明的实施方式，本领域技术人员可由本说明书所揭露的内容轻易地了解本发明的其他优点与功效。本发明还可以通过另外不同的具体实施方式加以实施或应用，本说明书中的各项细节也可以基于不同观点与应用，在没有背离本发明的精神下进行各种修饰或改变。需要说明的是，以下实施例中所提供的图示仅以示意方式说明本发明的基本构想，在不冲突的情况下，以下实施例及实施例中的特征可以相互组合。

[0032] 其中，附图仅用于示例性说明，表示的仅是示意图，而非实物图，不能理解为对本发明的限制；为了更好地说明本发明的实施例，附图某些部件会有省略、放大或缩小，并不代表实际产品的尺寸；对本领域技术人员来说，附图中某些公知结构及其说明可能省略是可以理解的。

[0033] 本发明实施例的附图中相同或相似的标号对应相同或相似的部件；在本发明的描述中，需要理解的是，若有术语“上”、“下”、“左”、“右”、“前”、“后”等指示的方位或位置关系为基于附图所示的方位或位置关系，仅是为了便于描述本发明和简化描述，而不是指示或暗示所指的装置或元件必须具有特定的方位、以特定的方位构造和操作，因此附图中描述位置关系的用语仅用于示例性说明，不能理解为对本发明的限制，对于本领域的普通技术人员而言，可以根据具体情况理解上述术语的具体含义。

[0034] 请参阅图1，为基于LSF的单主机多GPU分布式pytorch并行计算方法。

[0035] 编写LSF作业提交指令。

[0036] BSUB-q HPC.S1.GPU.X785.sha

[0037] 指定作业提交队列

[0038] #BSUB-n 8

[0039] 设定使用的核心数为8。同时表明本次任务将开启8个任务，使用8张GPU。

[0040] #BSUB-gpu"num＝4:mode＝exclusive_process"

[0041] 设定每台主机使用4张GPU，并且任务将独占分配的GPU。该语句将使得每台主机上环境变量CUDA_VISIBLE_DEVICES的值为‘0,1,2,3’，即有编号为0,1,2,3的GPU可供使用

[0042] #BSUB-o%J.out

[0043] #BSUB-e%J.err

[0044] 指定输出文件和错误文件。

[0045] python resnet_v_6.py

[0046] 作业提交语句。使作业能发送到主机上分布式计算。

[0047] 1、初始化进程组，由于使用GPU通信，后端应该写为NCCL。不过经过实验，即使错写为gloo，DDP内部也会自动使用NCCL作为通信模块。

[0048] 2、由于后面使用DDP包裹模型进行训练，其内部会自动将所有rank的模型权重同

步为rank 0的权重，因此只需在rank 0上读取模型权重即可。这是基于Pytorch版本1.12.1，低级版本似乎没有这个特性，需要在不同rank分别导入权重，则load需要传入map_location，如下面注释的两行代码所示。

[0049] 3、这里创建model的优化器，而不是创建用ddp包裹后的ddp_model的优化器，是为了兼容单GPU训练，读取优化器权重更方便。

[0050] 4、将优化器权重读取至该进程占用的GPU。如果没有map_location参数，load会将权重读取到原本保存它时的设备。

[0051] 5、优化器获取权重。经过实验，即使权重不在优化器所在的GPU，权重也会迁移过去而不会报错。当然load直接读取到相应GPU会减少数据传输。

[0052] 6、DDP包裹模型，为模型复制一个副本到相应GPU中。所有rank的模型副本会与rank0保持一致。注意，DDP并不复制模型优化器的副本，因此各进程的优化器需要在初始化时保持一致。权重要么不读取，要么都读取。

[0053] 7、这里开始模型的训练。数据需转移到相应的GPU设备。

[0054] 8、在backward中，所有进程的模型计算梯度后，会进行平均(不是相加)。也就是说，DDP在backward函数添加了hook，所有进程的模型梯度的ring_reduce将在这里执行。这个可以通过给各进程模型分别输入不同的数据进行验证，backward后这些模型有相同的梯度，且验算的确是所有进程梯度的平均。此外，还可以验证backward函数会阻断(block)各进程使用梯度，只有当所有进程都完成backward之后，各进程才能读取和使用梯度。这保证了所有进程在梯度上的一致性。

[0055] 9、各进程优化器使用梯度更新其模型副本权重。由于初始化时各进程模型、优化器权重一致，每次反向传播梯度也保持一致，则所有进程的模型在整个训练过程中都能保持一致。

[0056] 10、由于所有进程权重保持一致，只需通过一个进程保存即可。

[0057] 11、定义rank 0的IP和端口，使用mp.spawn，只需在主进程中定义即可，无需分别在子进程中定义。

[0058] 12、创建子进程，传入：子进程调用的函数(该函数第一个参数必须是rank)、子进程函数的参数(除了rank参数外)、子进程数、是否等待所有子进程创建完毕再开始执行。

[0059] 最后说明的是，以上实施例仅用以说明本发明的技术方案而非限制，尽管参照较佳实施例对本发明进行了详细说明，本领域的普通技术人员应当理解，可以对本发明的技术方案进行修改或者等同替换，而不脱离本技术方案的宗旨和范围，其均应涵盖在本发明的权利要求范围当中。

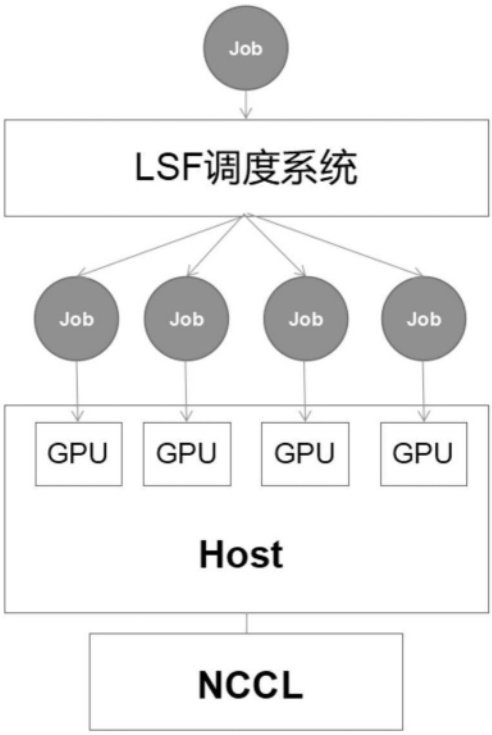


图1