



中国移动
China Mobile

九天平台分布式训练原理与实践

闫晓瑞、丛鹏宇 2023年3月

www.10086.cn

目 录

CONTENTS

- ▼ 分布式训练的架构
- ▼ 分布式训练的原理
- ▼ 分布式训练资源介绍
- ▼ 分布式训练的平台使用

数据并行和模型并行

- 数据并行是最常用的分布式训练策略，适用于大规模数据的训练
- 模型并行适用于超大规模的网络，层数太深或者参数太多

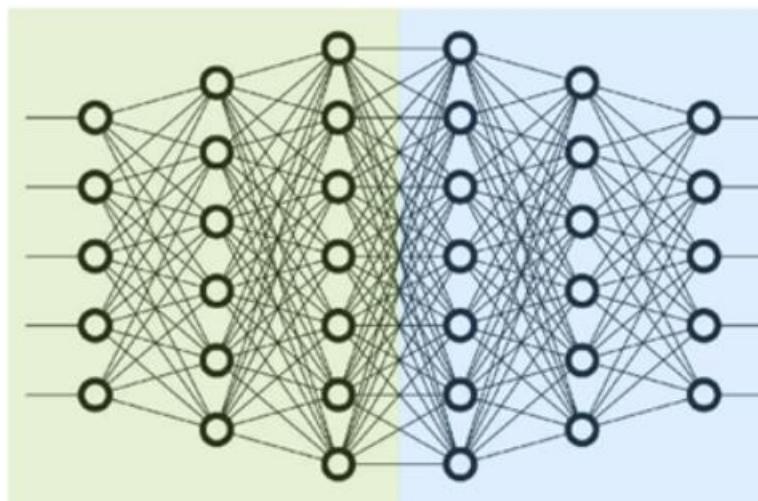
模型并行从计算图的切分角度，可以分为以下几种：

流水并行：按模型的layer层切分到不同设备，即层间并行。

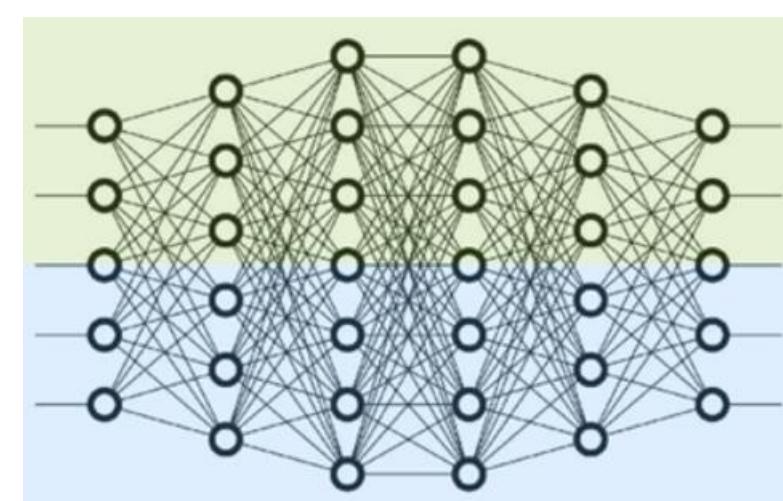
张量模型并行：将计算图中的层内的参数切分到不同设备，即层内并行。

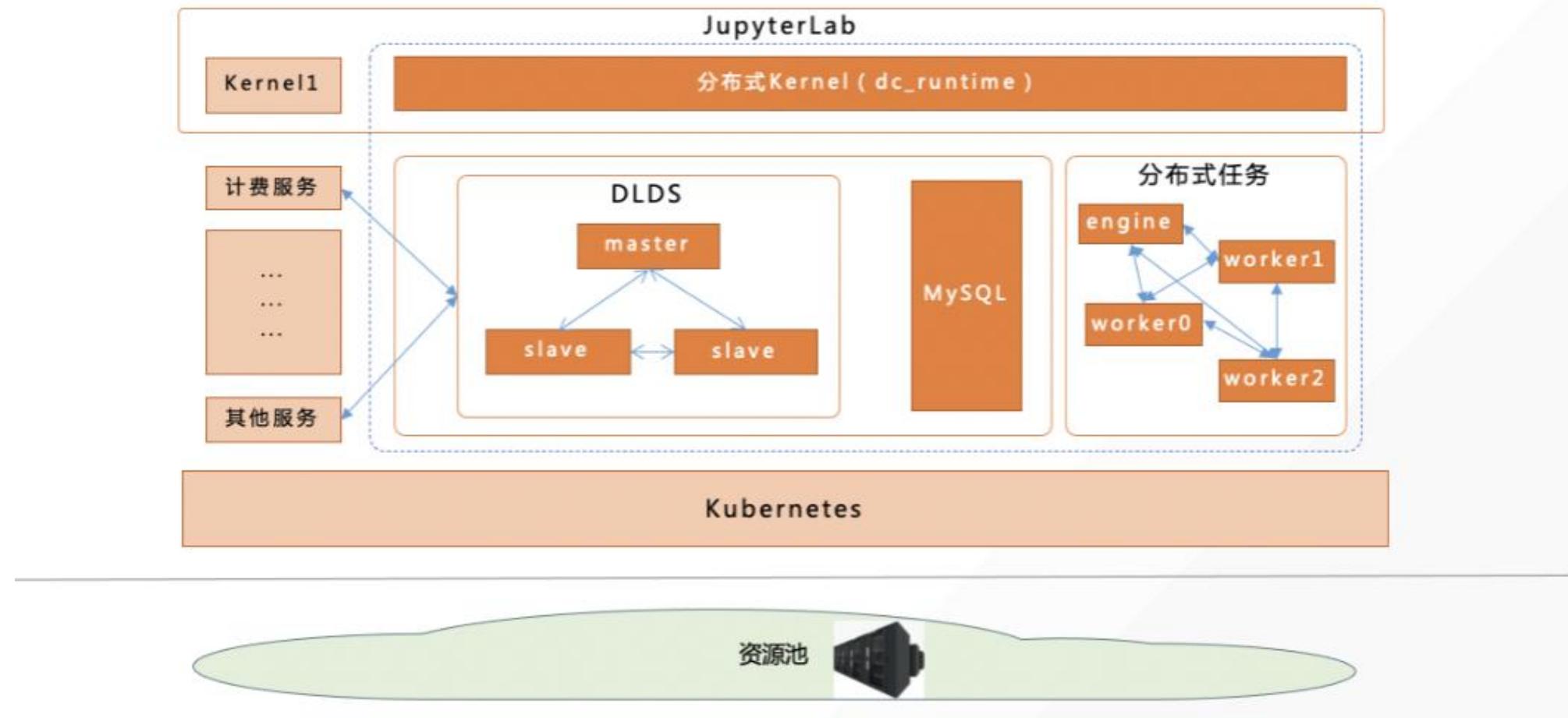
- 混合并行

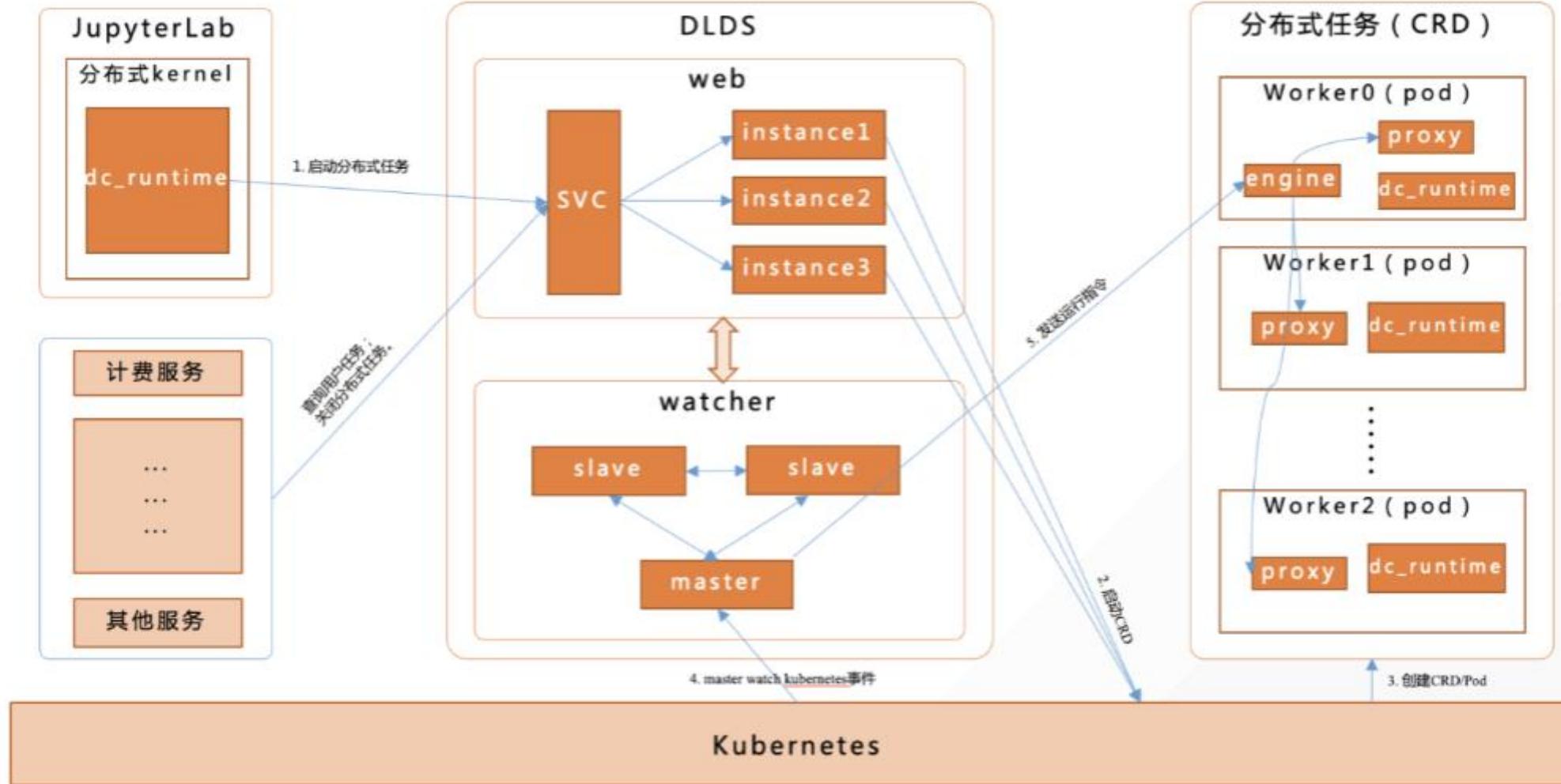
层间并行



层内并行









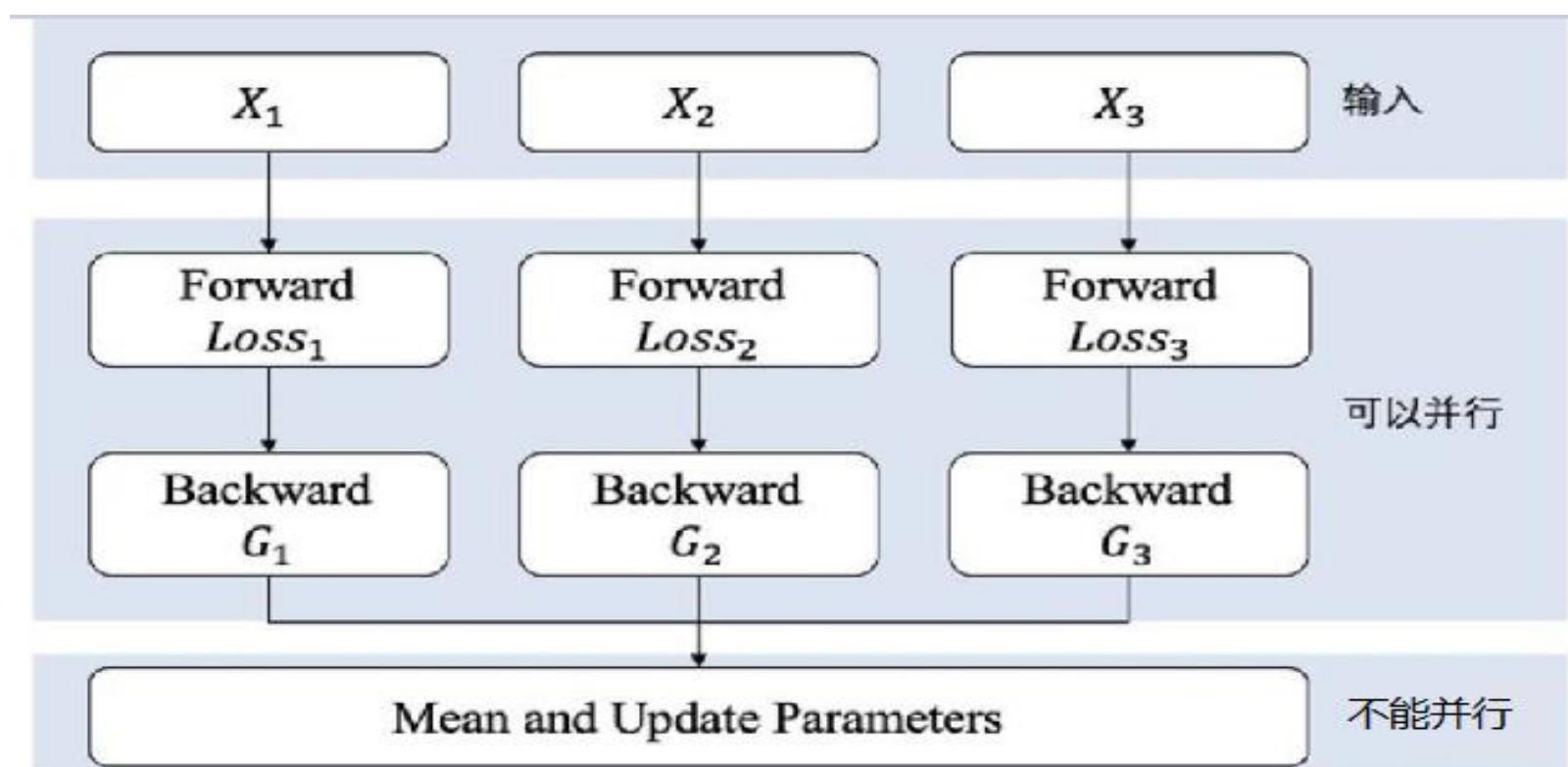
CONTENTS

- ▼ 分布式训练架构
- ▼ 分布式训练原理
- ▼ 分布式训练资源介绍
- ▼ 分布式训练平台使用

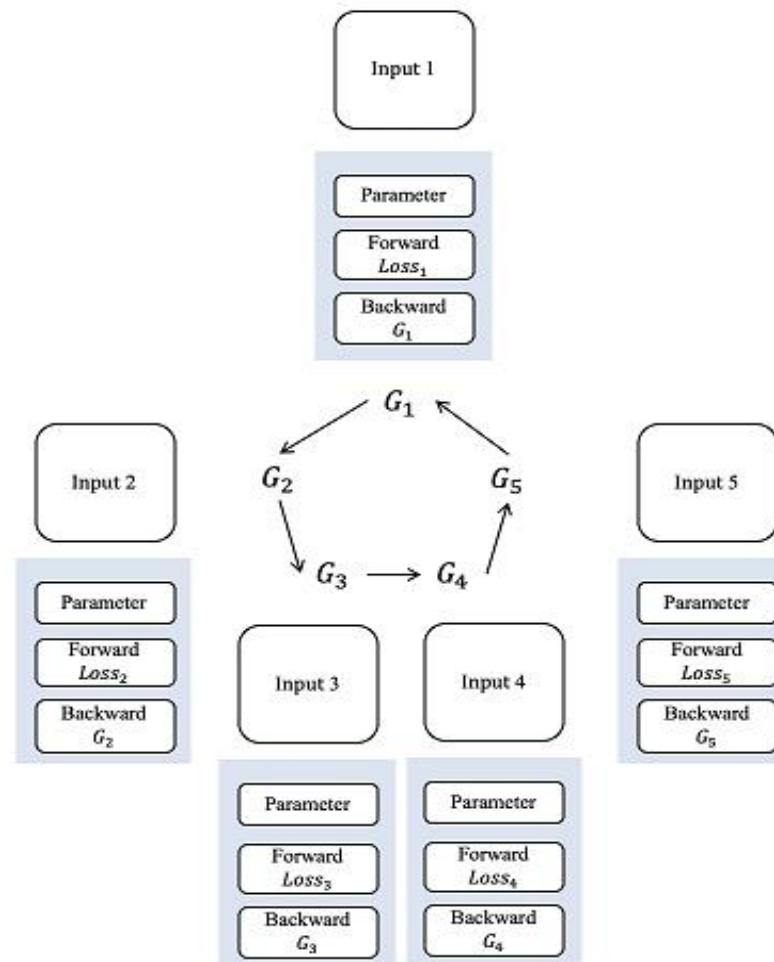
在训练神经网络模型时，我们更偏爱大规模的数据集和复杂的网络结构。更大规模的数据集和更复杂的网络结构可以让我们的模型表征能力更强，但同时也对计算的时间和空间提出了挑战。

如果我们只用单机单GPU来跑，则会出现一卡有难（运行时间长、显存不足等），十卡围观的状态。很多神经网络库都提供了分布式训练的API，但是如果不懂得内在机理，我们仍然很难得到满意的效率和效果。

首先需要明确的是神经网络模型的训练过程中，**哪些部分可以并行，哪些部分不能并行。**



Ring Allreduce: 在不使用中心节点的前提下完成 Map 和 Reduce。

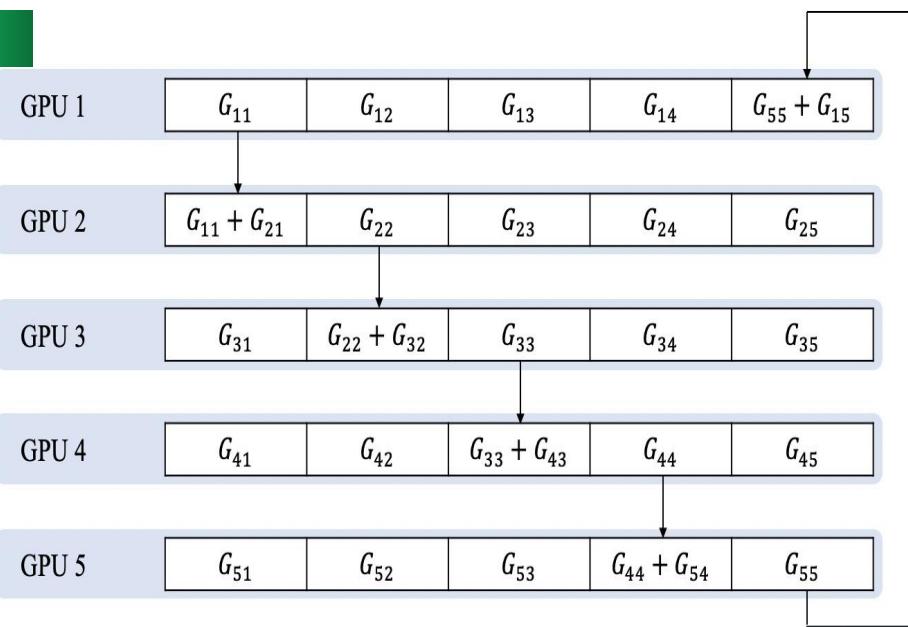


GPU 计算模型的梯度 G_i 时，根据模型的层数 j 将 G_i 分成 j 份。我们求 mean \bar{G} ，也就是在求每一个 mean (\bar{G}_j)。我们标记每份梯度为： G_{ij} 。对于一个 5 层的网络，我们可以写成这样：

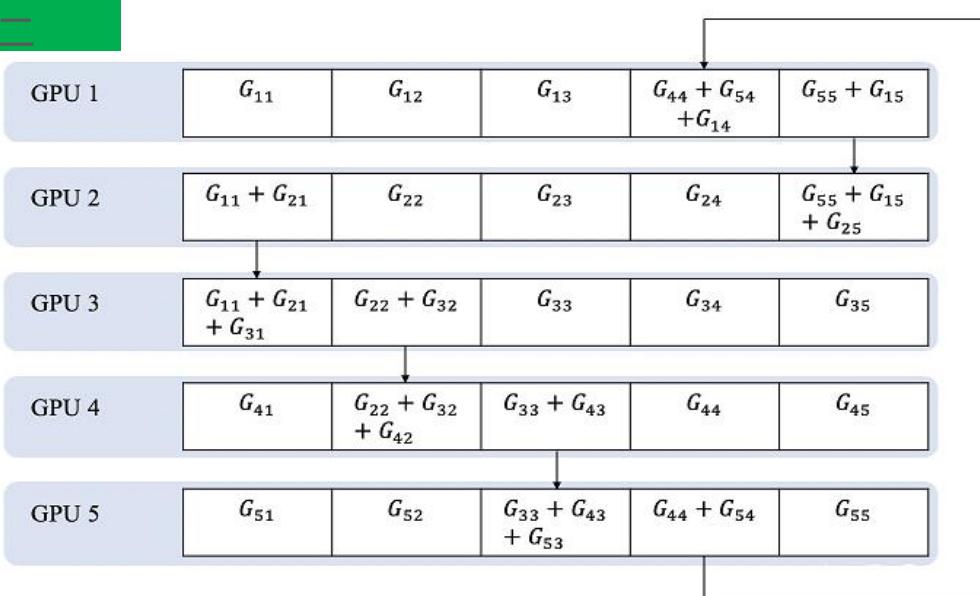
GPU 1	G_{11}	G_{12}	G_{13}	G_{14}	G_{15}
GPU 2	G_{21}	G_{22}	G_{23}	G_{24}	G_{25}
GPU 3	G_{31}	G_{32}	G_{33}	G_{34}	G_{35}
GPU 4	G_{41}	G_{42}	G_{43}	G_{44}	G_{45}
GPU 5	G_{51}	G_{52}	G_{53}	G_{54}	G_{55}

Ring AllReduce -- Scatter Reduce

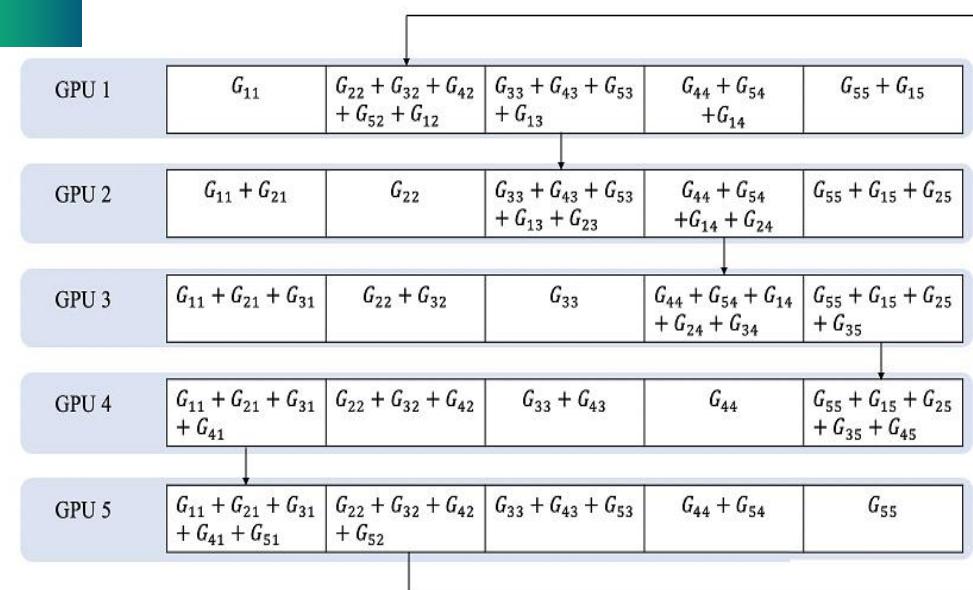
图一



图二

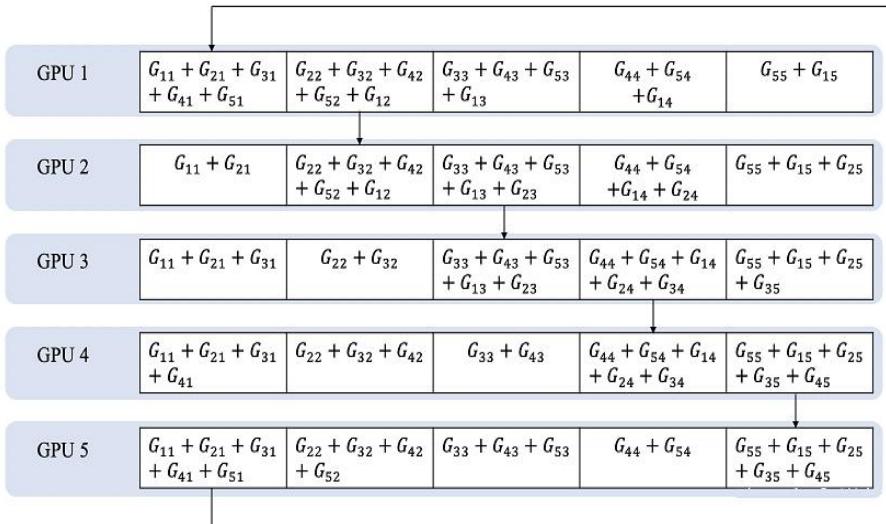


图四

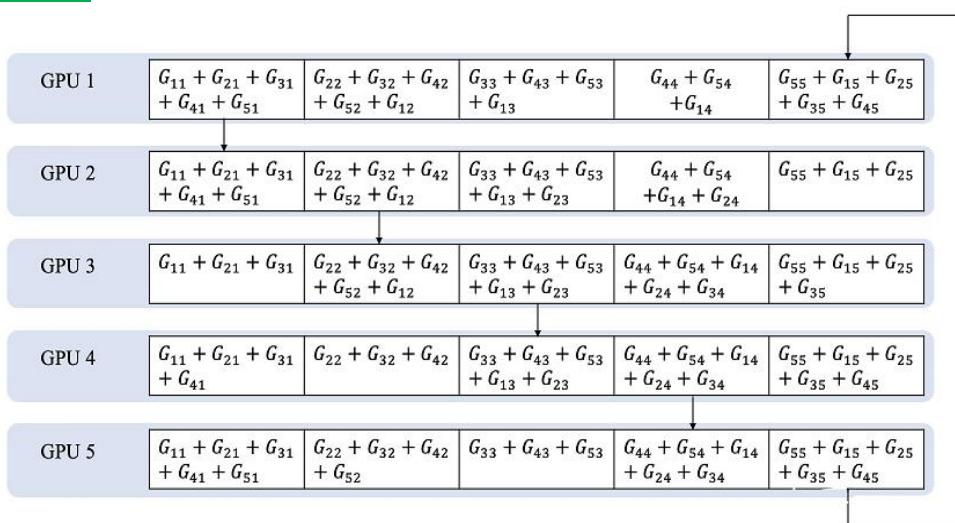


Ring AllReduce -- All Gather

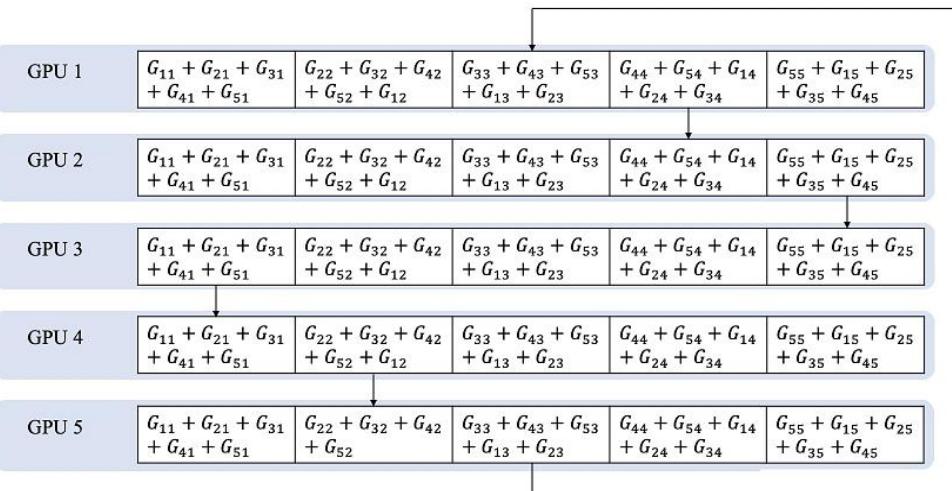
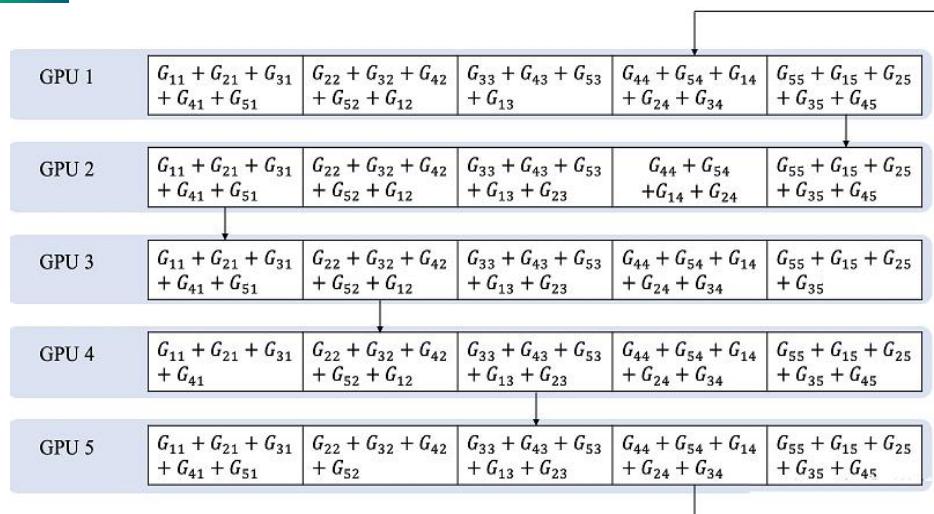
图五



图六



图七



AllReduce实际上是scatter reduce + all gather的过程。

reduce scatter: 每个GPU都保存了一部分规约的结果

all gather: 每个gpu都聚合了其他所有GPU的结果

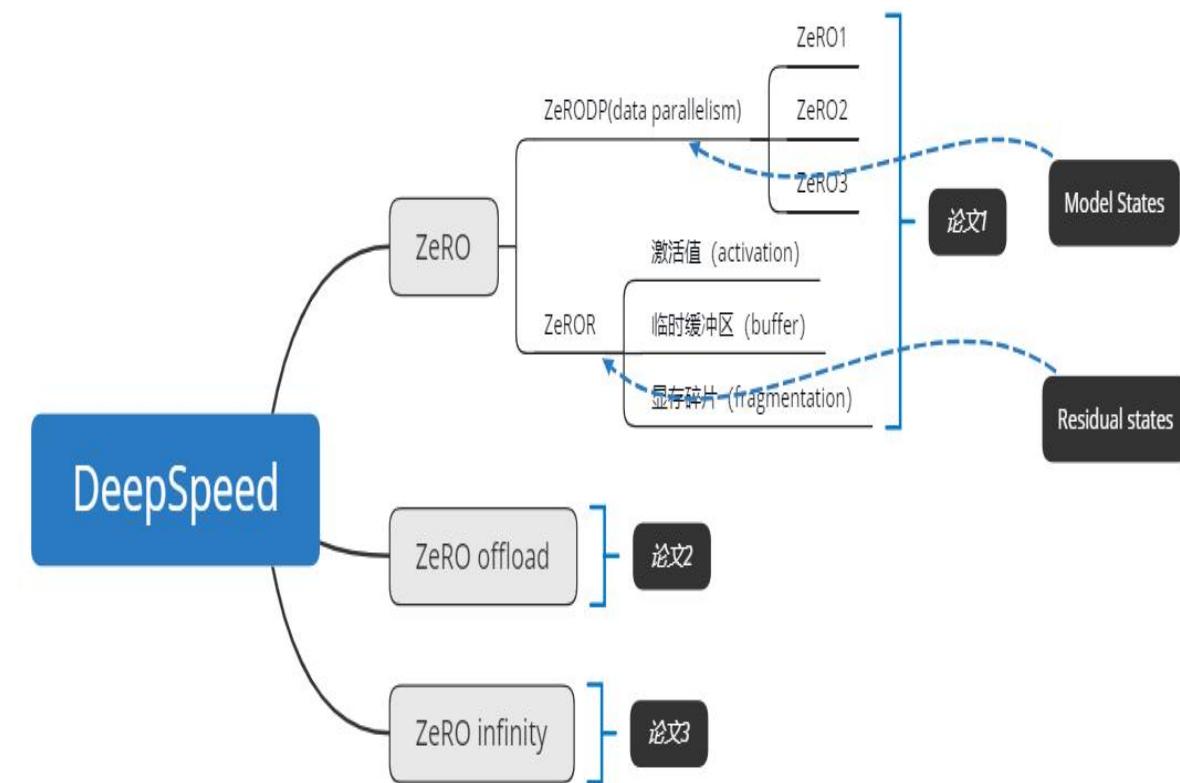
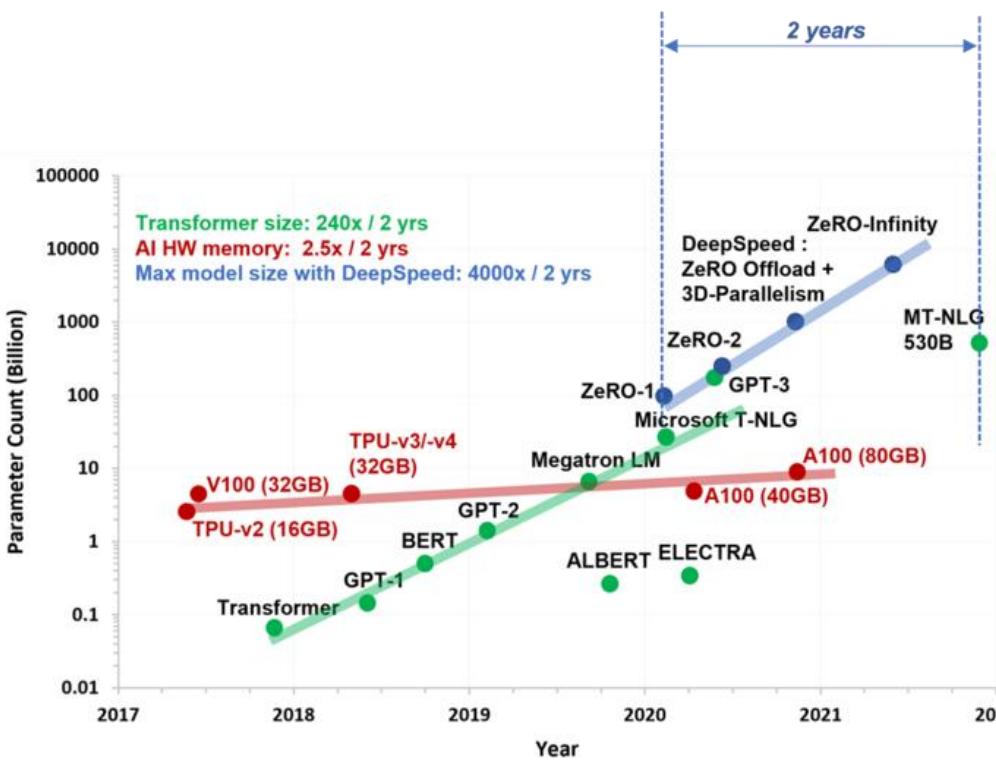
Ring AllReduce 经过前4步：传递、求和的过程，后4步：传递、覆盖的过程。共8次传输，完成参数更新。

最后只需要再除以 N (N 为 GPU 个数) 得到的结果就是 $\text{mean}(\bar{G})$ 了。

由此可见：利用 Ring Allreduce，然可以像 Parameter Server 一样在 $2*(N-1)$ 次传输后完成参数的更新过程。而且每次传输不再是针对中心节点，而是分散在各节点的两两之间，大大减小了对带宽的压力，实现了带宽并行。

实现过程见下面：分布式训练平台使用

- Zero Redundancy Optimizer (ZeRO)
- 显卡资源有限加载和训练大模



DeepSpeed是一个开源深度学习训练优化库，它的显存优化技术——ZeRO（零冗余优化器），它通过将模型参数拆散分布到各个GPU上，以实现大型模型的计算，这也就意味着我们能用更少的GPU训练更大的模型，而且不受限于显存。

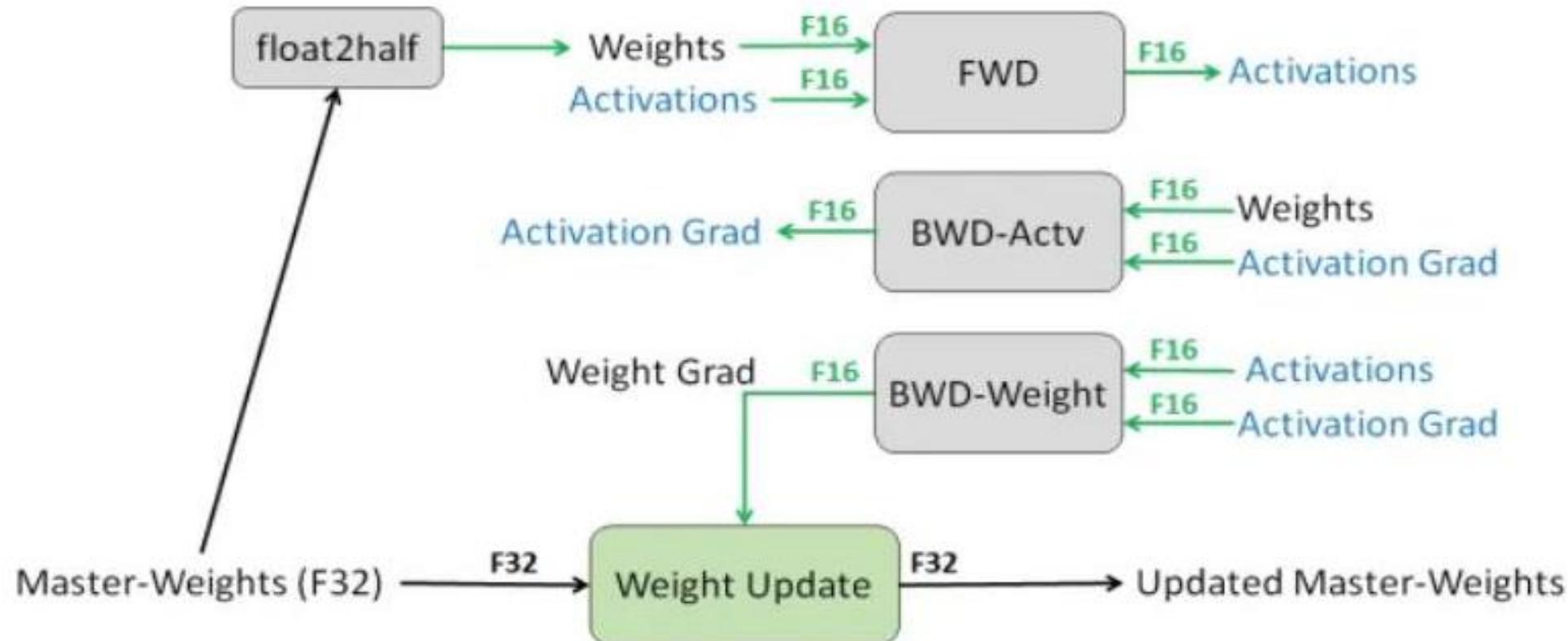
Deep Speed + Zero



大规模训练中的显存占用可以分为 Model States 与 Activation 两部分，而 ZeRO 就是为了解决 Model States 而诞生的一项技术。

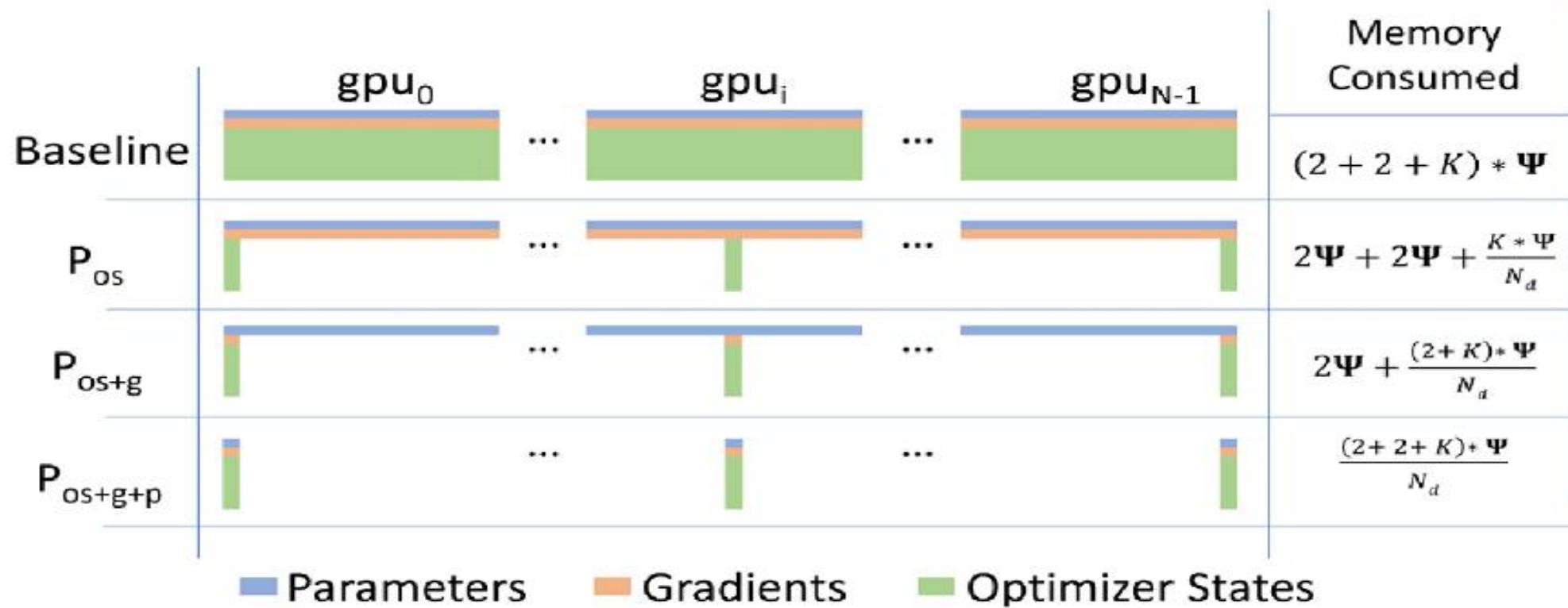
ZeRO 的本质，是在数据并行的基础上，对冗余空间占用进行深度优化。ZeRO 有三个不同级别，分别对应对 Model States 不同程度的分割 (Partition)：

- ZeRO-1：分割 Optimizer States；
- ZeRO-2：分割 Optimizer States 与 Gradients；
- ZeRO-3：分割 Optimizer States、Gradients 与 Parameters；

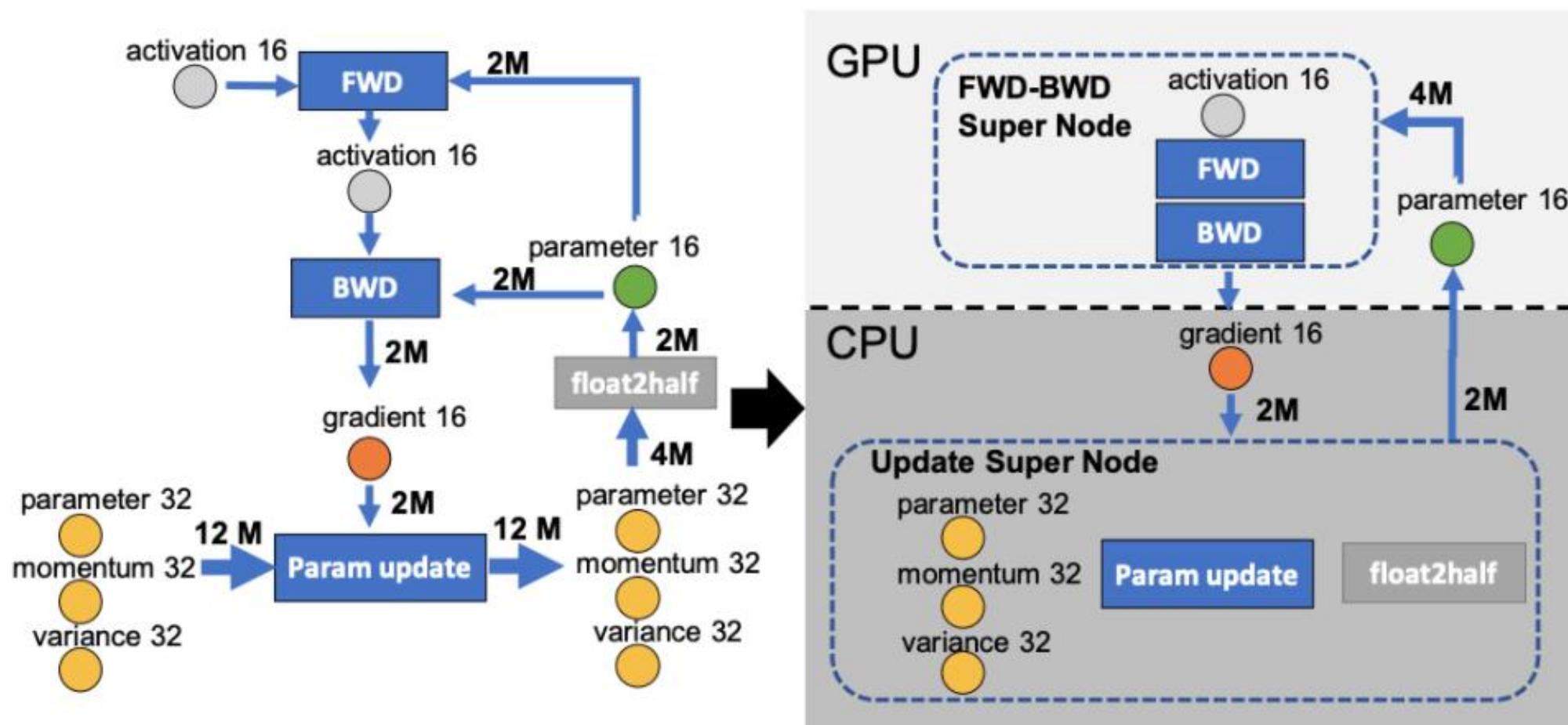


Mixed precision training iteration for a layer.

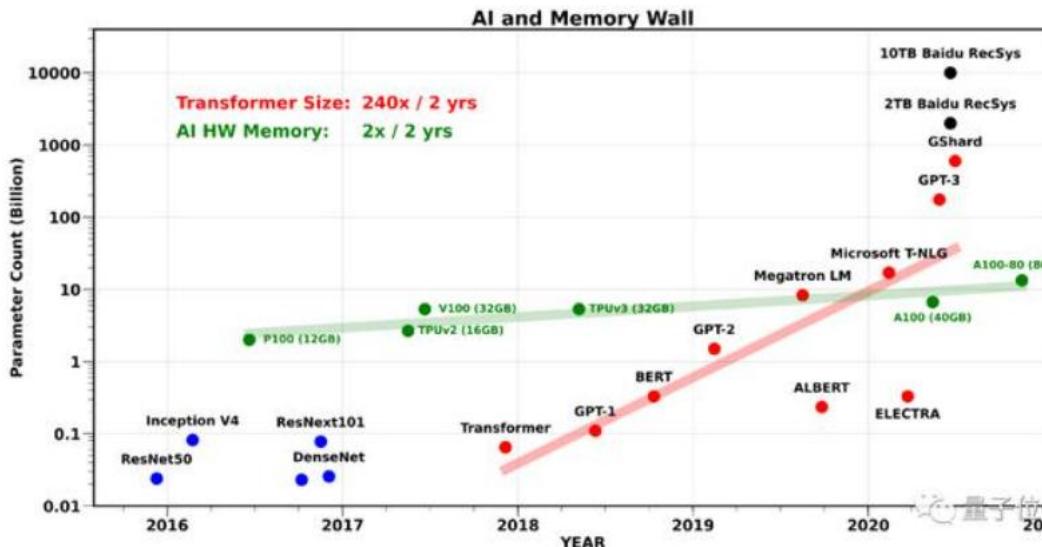
ZeRO-DP的三个阶段



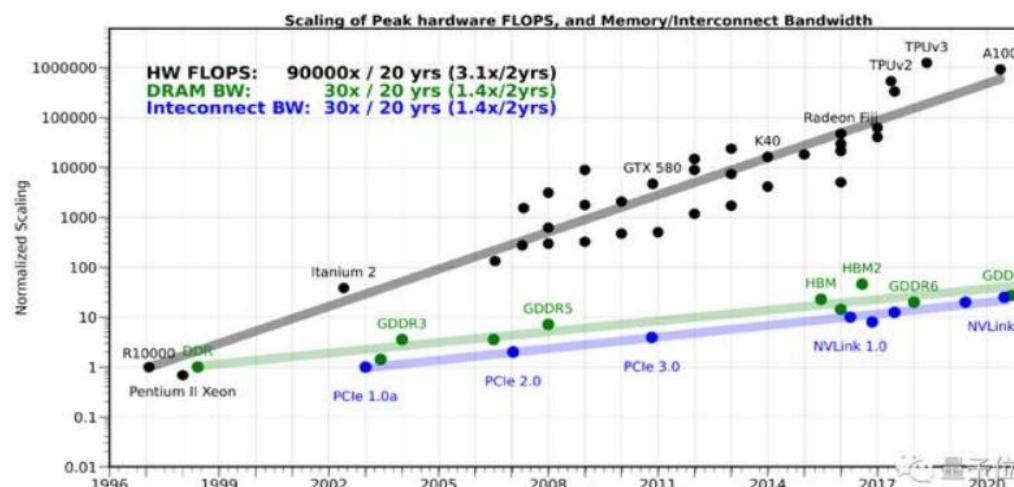
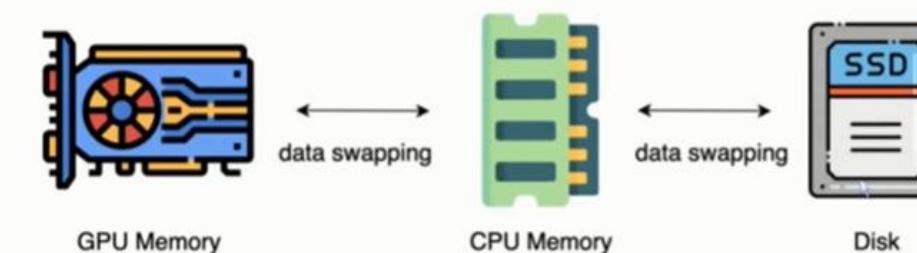
ZeRO offload



ZeRO infinity



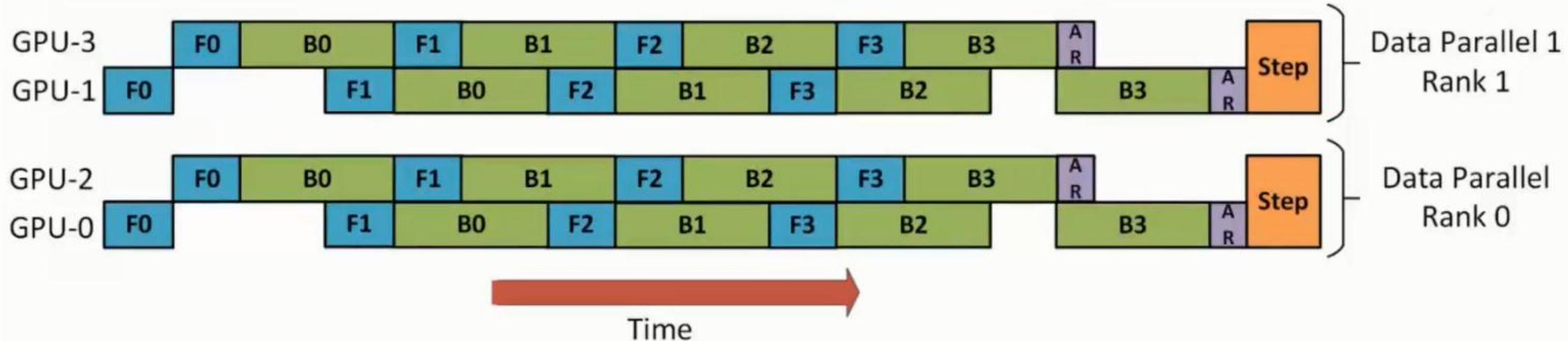
ZeRO-Infinity



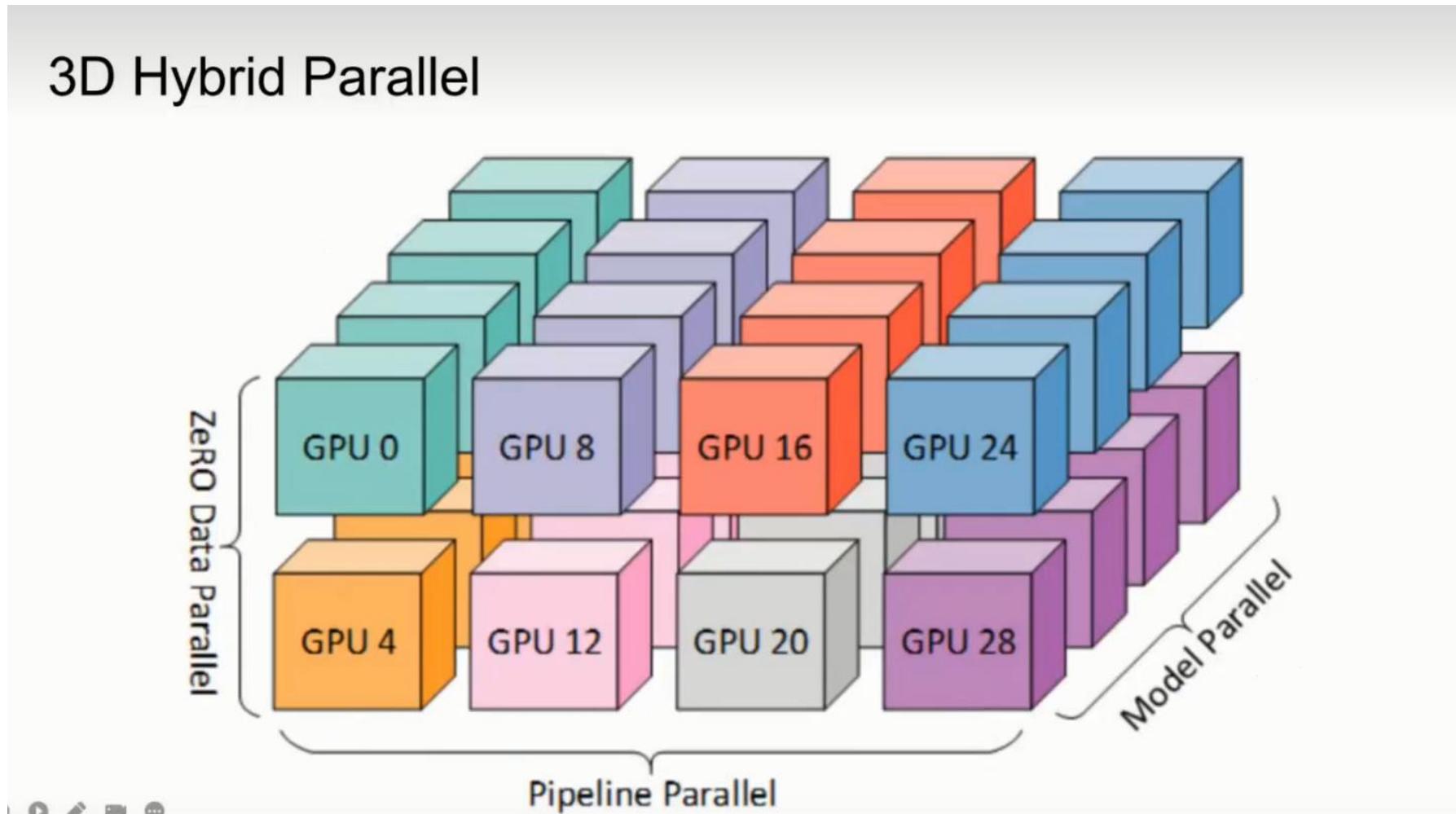
内存墙问题：Transformer模型的参数量增长了240倍，而同期硬件的显存（V100 32G A100 80G）只增长了2倍；过去20年，硬件的峰值计算提升了90000倍，但是内存和硬件之间的带宽仅仅提升了30倍。**Zero Infinity**是为了解决内存墙问题的，用**NVME**接口打通了内存跟ssd固态硬盘之间的

Deepspeed 流水线并行

通过不同的Group维护数据并行组，模型并行组和流水线并行组。其中dp group做allreduce同步梯度，同一个pipe group通过broadcast去send activation（前向）或者send grads（反向）。同一个pipe group只有pipeline的第一个stage需要读取train数据，pipeline的最后一个stage需要读取label数据，算出loss。



Deepspeed 3D混合并行





CONTENTS

- ▼ 分布式训练架构
- ▼ 分布式训练原理
- ▼ 分布式训练资源介绍
- ▼ 分布式训练平台使用

资源名称	作用	数量	个人额度	规格	备注	
engine	jupyter客户端	300个	运行中4个	1C8G	cpu实例	
GPU	训练卡数	128	6	4C32G/gpu	限额80	
网络	IB			100Gb		

默认支持tf2.1和pytorch1.5，也支持用户自定tf2和pytorch环境

新加入了deepspeed-dl 多机多卡分布式并行及示例



CONTENTS

- ▼ 分布式训练架构
- ▼ 分布式训练原理
- ▼ 分布式训练资源介绍
- ▼ 分布式训练平台使用

九天深度学习分布式训练使用

使用路径: <http://jiutian.hq.cmcc>

OA登录，选择九天-深度学习平台 V3.0

新建分布式训练实例

模型训练模块，新建分布式训练实例

分布式训练实例为1C8G的CPU实例，因此：

- a. 不能用于普通的gpu训练(没有分布式需求的可以创建gpu实例)

分布式实例中资源不能用于普通的gpu训练(对于非分布式需求，可以发起单卡任务，或者使用一般gpu实例)

- b. 如果安装类似人脸检测库dlib，内存超限崩掉，可联系我们手动解决，也可以先创建gpu实例，安装好环境，copy到分布式训练实例。

分布式训练的实例的好处

- a. 分布式训练实例，不会定期回收，除非资源耗尽
- b. 分布式训练实例的flash,share, teamshare目录，可以直接作为所有数据目录共享目录。也可小批量做闪存支持
- c. 配额最多可达80卡，缩小大模型、大数据量的训练时间
- d. gpu资源随时使用，用完回收，节约集群资源

权限和额度申请

分布式训练支撑大模型和大数据量的训练，由于卡资源和存储资源紧张，故使用需要申请开通相应权限。

1. 联系文卉开通九天平台分布式训练使用权限，创建1C8G cpu jupyter client
2. 默认gpu额度为6卡，注：卡资源共享，非独占；资源紧张需要抢资源
3. 小批量调试代码，调通后可根据需要联系文卉，申请调高卡额度

注： 分布式训练实例为cpu实例，不挂载gpu，只有训练任务启动才会动态挂载gpu，用完释放
分布式gpu额度共享，非独享，无法保证每个用户都能启动到自己的额度值。

构建环境

参考jupyter实例中: /root/分布式任务使用说明 (Terminal版) .ipynb

如何自定义环境中进行任务定义和任务运行

1. 在 terminal 中使用 conda 命令自定义深度学习分布式环境

打开 terminal 窗口， 执行 conda env create -n \$env名称 -f /opt/conda/dl_env_config.yaml 命令， 新建环境

2. 执行 conda activate \$env名称 命令， 激活环境

3. 执行 python -m ipykernel install --profile=dc --user --name \$kernel名称 命令， 新建 kernel

4. 安装所需第三方包，例如：

```
pip install tensorflow-gpu==2.1.0 tensorflow-text==2.1.1 --no-deps tensorflow-hub==0.11.0 tf-models-official==2.1.0.dev1
```

5. 在自定义环境中，进行分布式任务定义和任务运行

```
export DC_DL_CONFIG='{"framework":"pytorch","worker": {"count":2,"gpus":1}}'
```

```
python dc_dl_run.py --name=test --file=run.py --queue_time=10 --nohup=true
```

代码改造

a. 输入参数，在入口.py中定义，例如：

```
def get_args():

    parser = argparse.ArgumentParser(description='PyTorch ImageNet Training')
    parser.add_argument('--data', metavar='DIR', default='./data', help='path to dataset')
    parser.add_argument('-b', '--batch_size_p_n', default=256, type=int,
                        metavar='N',
                        help='mini-batch size (default: 256), this is the total '
                             'batch size of all GPUs on the current node when '
                             'using Data Parallel or Distributed Data Parallel')
    parser.add_argument('-j', '--workers', default=4, type=int, metavar='N',
                        help='number of data loading workers (default: 4)')
    parser.add_argument('--epochs', default=6, type=int, metavar='N',
                        help='number of total epochs to run')
```

代码改造

b. `MASTER_ADDR`、`MASTER_PORT`、`WORLD_SIZE` 都直接从worker的环境变量中获取，不需要自己设置。

在`main()`函数中使用`torch.multiprocessing.spawn`进行分布式训练

```
def main():
    args = get_args()
    args.gpus = torch.cuda.device_count()
    args.nodes = int(os.environ.get("WORLD_SIZE", 1))
    print("WORLD_SIZE: ", os.environ['WORLD_SIZE'])
    args.world_size = args.gpus * args.nodes
    print("MASTER_ADDR: ", os.environ['MASTER_ADDR'])
    print("MASTER_PORT: ", os.environ['MASTER_PORT'])
    pprint.pprint(args)
    mp.spawn(train, nprocs=args.gpus, args=(args,))
```

代码改造

c. 在train函数中，初始化进程组：

```
torch.distributed.init_process_group(backend='nccl', init_method='env://', world_size=args.world_size, rank=args.rank)
```

注意，backend后端通信协议选择： 使用gpu训练时设置为“nccl”， 使用cpu训练时， 设置为“gloo”；

init_method初始化方式注意： 要使用'env://'方式， 不是tcp方式

```
ngpus_per_node = torch.cuda.device_count() #每个worker的gpus的数量
args.rank = int(os.environ.get("RANK"))*ngpus_per_node+gpu #分配rank号
print("os.environ.RANK: ", int(os.environ.get("RANK")))
print("args.rank: ", args.rank)
torch.distributed.init_process_group(backend='nccl', init_method='env://', world_size=args.world_size, rank=args.rank)
```

代码改造

d. 模型分发： 创建分布式并行模型 `model = torch.nn.parallel.DistributedDataParallel(model, device_ids=[gpu])`

```
model = ConvNet()
torch.cuda.set_device(gpu)
model.cuda(gpu) # 先传到GPU，然后再包装
model = nn.parallel.DistributedDataParallel(model, device_ids=[gpu])
```

e. 数据集创建Sampler：

由于 `DistributedSampler` 在实例化sampler的时候就已经shuffle过数据了（`DistributedSampler`默认`shuffle=true`），所以在定义dataloader的时候不需要再将shuffle设置为True了
(当`shuffle=True`时，用于random的随机数种子。这个参数在所有分布式进程中必须保持一致，默认为0)

代码改造

```
# Data loading code
train_dataset = torchvision.datasets.MNIST(root='./data',
                                             train=True,
                                             transform=transforms.ToTensor(),
                                             download=False)

train_sampler = torch.utils.data.distributed.DistributedSampler(train_dataset,
                                                               num_replicas=args.world_size,
                                                               rank=args.rank)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=False,
                                           num_workers=0,
                                           pin_memory=True,
                                           sampler=train_sampler)
```

代码改造

f. 定义好optimizer和loss func，就可以开始训练了

```
# define loss function (criterion) and optimizer
criterion = nn.CrossEntropyLoss().cuda(gpu)
optimizer = torch.optim.SGD(model.parameters(), args.lr,
                           momentum=args.momentum,
                           weight_decay=args.weight_decay)
```

.....

代码改造

```
for epoch in range(args.epochs):
    trainloader.sampler.set_epoch(epoch)
    for i, (images, labels) in enumerate(train_loader):
        images = images.cuda(non_blocking=True)
        labels = labels.cuda(non_blocking=True)
        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)
        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if (i + 1) % 10 == 0 and gpu == 0:
            print('rank{}, Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'.format(dist.get_rank(),
                epoch + 1, args.epochs, i + 1, total_step, loss.item()))
```

代码改造

g. 训练过程中，监控gpu和mem使用情况，可以打印log或者存log出来，

```
for epoch in range(args.epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.cuda(non_blocking=True)
        labels = labels.cuda(non_blocking=True)
        outputs = model(images)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if (i + 1) % 20 == 0 and gpu == 0:
            ###监控gpu和显存的使用情况
            print("os.popen---nvidia-smi:\n", os.popen("nvidia-smi").read())
```

环境搭建

1. 进入分布式训练实例，挂载pile、等相应的数据集
2. 选择deepspeed-dl 分布式并行 kernel
3. deepspeed-dl内置gpt-neox代码，可以试运行，后面以gpt-neox为例讲解使用方法

```
(deepspeed-dl) root@d1-230302104630317-pod-jupyter-565d7649f7-k2px:~/flash/yanxr/gpt-neox/code/gpt-neox# conda env list
# conda environments:
#
base                  /opt/conda
deep_learning          /opt/conda/envs/deep_learning
deepspeed-dl           * /opt/conda/envs/deepspeed-dl
julia                 /opt/conda/envs/julia
oneflow0.8             /opt/conda/envs/oneflow0.8
python2                /opt/conda/envs/python2
pytorch1.8              /opt/conda/envs/pytorch1.8
r                      /opt/conda/envs/r
rapids                 /opt/conda/envs/rapids
tf1.14                 /opt/conda/envs/tf1.14
tf2.4.1                /opt/conda/envs/tf2.4.1
```

使用入口文件ds_run.py

```
slots=str(torch.cuda.device_count())
workers=os.environ.get("ALL_WORKERS")
hosts=workers.replace(' ', '').split(',')
print(hosts)
print("host1=", hosts[1])
hostfiles=""
for host in hosts:
    host=host.strip('\'')
    hostfiles += host + " slots=" + slots + "\n"
print(hostfiles)
fileName='hostfilexr'
with open(fileName, 'w')as file:
    file.write(hostfiles)
```

使用入口文件ds_run.py

1. 组装启动命令cmd，并在rank0中启动：

```
#cmd="export DLTS_HOSTFILE=hostfilexr ; /root/.local/conda/envs/deepspeed-gpt/bin \
#/python ./deepy.py train.py configs/19M.yml"
cmd="export DLTS_HOSTFILE=hostfilexr ; /opt/conda/envs/deepspeed-dl/bin/python \
./deepy.py train.py ./configs/6-7B.yml ./configs/local_setup.yml"
rank=int(os.environ['RANK'])
if rank==0:
    os.system(cmd)
```

开始训练

1. 设置资源:

```
(deepspeed-gpt) root@d1-230302104630317-pod-jupyter-565d7649f7-k2pxn:~/flash/yanxr/gpt-neox/code/gpt-neox# export DC_DL_CONFIG='{"framework":"pytorch","worker":{"count":4,"gpus":8 }}'
```

2. python命令开始训练:

```
(deepspeed-d1) root@d1-230314120232fcw-pod-jupyter-96559d7d5-srdt6:~/flash/yanxr/gpt-neox/code/gpt-neox# python dc_dl_run.py --name=gpt-neox --file=ds_run.py --queue_time=10
=====dc_dl_config===== {"framework":"pytorch","worker": {"count":2,"gpus":8 }}gpt-neox/code/gpt-neox# python dc_dl_run.py --name=gpt-neox --file=ds_run.py --queue_time=10
/opt/conda/envs/deepspeed-d1/lib/python3.8/site-packages/requests/_init_.py:102: RequestsDependencyWarning: urllib3 (1.26.9) or chardet (5.1.0)/charset_normalizer (2.0.4) doesn't match a supported version!
  warnings.warn("urllib3 ({}) or chardet ({})/charset_normalizer ({}) doesn't match a supported
"
```

查看运行状态:

查看运行状态：

```
2023-03-15 13:21:52.800 [INFO] 10.42.3.208: steps: 20 loss: 11.0396 iter time (s): 5.658 samples/sec: 1.414
2023-03-15 13:21:52.800 [INFO] 10.42.3.208: %comms: 76.25213116021334
2023-03-15 13:21:52.800 [INFO] 10.42.3.208: %optimizer_step 0.6853222299409336
2023-03-15 13:21:52.800 [INFO] 10.42.3.208: %forward: 6.236562757469278
2023-03-15 13:21:52.800 [INFO] 10.42.3.208: %backward: 17.15262592093925
2023-03-15 13:21:52.800 [INFO] 10.42.3.208: [2023-03-15 13:21:52,800] [INFO] [logging.py:60:log_dist] [Rank 0] rank=0 time (ms) | train_batch: 0.00 | batch_input: 8.58 | forward: 3528.52 | pipe_send_output: 83.28 | comms: 43141.87 | pipe_recv_grad: 40922.54 | backward: 9704.60 | reduce_tied_grads: 0.24 | reduce_grads: 1929.67 | step: 387.74 | _step_clipping: 0.09 | _step_step: 386.42 | _step_zero_grad: 0.36 | _step_check_overflow: 0.34
2023-03-15 13:22:50.727 [INFO] 10.42.3.208: [2023-03-15 13:22:50,727] [INFO] [logging.py:60:log_dist] [Rank 0] step=30, skipped=18, lr=[4.500000000000000e-07, 4.5000000000000003e-07], mom=[[0.9, 0.95], [0.9, 0.95]]
2023-03-15 13:22:50.728 [INFO] 10.42.3.208: steps: 30 loss: 9.2521 iter time (s): 5.792 samples/sec: 1.381
2023-03-15 13:22:50.728 [INFO] 10.42.3.208: %comms: 76.03927282562202
2023-03-15 13:22:50.728 [INFO] 10.42.3.208: %optimizer_step 2.859155581949987
2023-03-15 13:22:50.728 [INFO] 10.42.3.208: %forward: 6.0733241917818575
2023-03-15 13:22:50.728 [INFO] 10.42.3.208: %backward: 16.761201154570387
2023-03-15 13:22:50.728 [INFO] 10.42.3.208: [2023-03-15 13:22:50,728] [INFO] [logging.py:60:log_dist] [Rank 0] rank=0 time (ms) | train_batch: 0.00 | batch_input: 9.04 | forward: 3517.88 | pipe_send_output: 93.22 | comms: 44044.57 | pipe_recv_grad: 40996.54 | backward: 9708.67 | reduce_tied_grads: 0.25 | reduce_grads: 1930.56 | step: 1656.12 | _step_clipping: 0.09 | _step_step: 1654.67 | _step_zero_grad: 0.32 | _step_check_overflow: 0.49
2023-03-15 13:23:48.701 [INFO] 10.42.3.208: [2023-03-15 13:23:48,701] [INFO] [logging.py:60:log_dist] [Rank 0] step=40, skipped=18, lr=[8.25e-07, 8.25e-07], mom=[[0.9, 0.95], [0.9, 0.95]]
2023-03-15 13:23:48.702 [INFO] 10.42.3.208: steps: 40 loss: 9.1411 iter time (s): 5.797 samples/sec: 1.380
2023-03-15 13:23:48.702 [INFO] 10.42.3.208: %comms: 76.06068860630491
2023-03-15 13:23:48.702 [INFO] 10.42.3.208: %optimizer_step 2.8678686065712506
2023-03-15 13:23:48.702 [INFO] 10.42.3.208: %forward: 6.059640904569079
2023-03-15 13:23:48.702 [INFO] 10.42.3.208: %backward: 16.749641458073437
2023-03-15 13:23:48.702 [INFO] 10.42.3.208: [2023-03-15 13:23:48,701] [INFO] [logging.py:60:log_dist] [Rank 0] rank=0 time (ms) | train_batch: 0.00 | batch_input: 8.92 | forward: 3512.74 | pipe_send_output: 84.89 | comms: 44092.00 | pipe_recv_grad: 41038.60 | backward: 9709.68 | reduce_tied_grads: 0.25 | reduce_grads: 1941.77 | step: 1662.49 | _step_clipping: 0.09 | _step_step: 1661.06 | _step_zero_grad: 0.31 | _step_check_overflow: 0.48
```

创建自定义环境：

1. 创建分布式基础环境：

```
cp -rf /opt/conda/dl_env_config.yaml dl_env_config.yaml
```

2. 修改所需要python或者特殊包的版本：

由于gpt-neox等deepspeed项目对某些库有特殊的需求，

```
if version.parse(platform.python_version()) < version.parse("3.7"):
    raise ImportWarning(
        "To use `datasets`, Python>=3.7 is required, and the current version of Python doesn't match this condition."
    )

if version.parse(pyarrow.__version__).major < 6:
    raise ImportWarning(
        "To use `datasets`, the module `pyarrow>=6.0.0` is required, and the current version of `pyarrow` doesn't match thi
    ion.\n"
    )
    "If you are running this in a Google Colab, you should probably just restart the runtime to use the right version o
ow."
)
```

截图(Alt + A)

创建自定义环境：

```
vi dl_env_config.yaml
```

修改所需要python或者特殊包的版本：

```
dependencies:  
  - python=3.8  
  - requests>=2.18.4  
  - ipykernel>=5.3.0  
  - nbformat>=5.0.7  
  - scikit-learn>=0.20.3  
  - pandas>=0.24.2  
  - h5py>=2.7.0  
  - pyarrow>=0.13.0  
~
```

3. 创建环境，安装所需要包：

```
conda env create -n $env名称 -f /opt/conda/dl_env_config.yaml
```

4. conda activate，然后pip install -r requirement.txt 安装第三方包

改造入口文件ds_run.py

1. 把系统内置的ds_run.py文件copy到自己的执行目录下:

```
cp /root/ds_run.py ds_run.py
```

2. 打开 ds_run.py文件，修改cmd执行

```
cmd="/root/.local/conda/envs/deepspeed-dl/bin/deepspeed --hostfile=hostfilex train.py -p 2 --step=20000 --deepspeed config=ds config.json"
print(cmd)
rank=int(os.environ['RANK'])
print("=====rank=====", rank)
if rank==0:
```

3. 激活自己的环境，设置资源，开始训练

```
export DC_DL_CONFIG='{"framework":"pytorch","worker": {"count":2,"gpus":8}}'
python dc_dl_run.py --name=test --file=ds_run.py --queue_time=10
```

查看训练过程：

以官网流水并行为例：

```
2023-03-15 13:50:07.611 [INFO] 10.42.3.209: Building extension module utils...
2023-03-15 13:50:07.611 [INFO] 10.42.3.209: Allowing ninja to set a default number of workers... (overridable by setting the environment variable MAX_JOBS=N)
2023-03-15 13:50:19.973 [INFO] 10.42.3.209: [1/2] c++ -MMD -MF flatten_unflatten.o.d -DTORCH_EXTENSION_NAME=utils -DTORCH_API_INCLUDE_EXTENSION_H -DPYTHON3_7_COMPILER_TYPE=\"_gcc\" -DPYTHON3_7_STDLIB=\"_libstdcpp\" -DPYTHON3_7_BUILD_ABI=\"_cxxabi1011\" -isystem /root/.local/conda/envs/deepspeed-dl/lib/python3.7/site-packages/torch/include -isystem /root/.local/conda/envs/deepspeed-dl/lib/python3.7/site-packages/torch/include/torch/csrc/api/include -isystem /root/.local/conda/envs/deepspeed-dl/lib/python3.7/site-packages/torch/include/TH -isystem /root/.local/conda/envs/deepspeed-dl/lib/python3.7/site-packages/torch/include/THC -isystem /root/.local/conda/envs/deepspeed-dl/include/python3.7m -D_GLIBCXX_USE_CXX11_ABI=0 -fPIC -std=c++14 -c /root/.local/conda/envs/deepspeed-dl/lib/python3.7/site-packages/deepspeed/ops/csrc/utils/flatten_unflatten.cpp -o flatten_unflatten.o
2023-03-15 13:50:20.275 [INFO] 10.42.3.209: [2/2] c++ flatten_unflatten.o -shared -L/root/.local/conda/envs/deepspeed-dl/lib/python3.7/site-packages/torch/lib -lc10 -ltorch_cpu -ltorch -ltorch_python -o utils.so
2023-03-15 13:50:20.280 [INFO] 10.42.3.209: Loading extension module utils...
2023-03-15 13:50:20.280 [INFO] 10.42.3.209: Time to load utils op: 12.984493970870972 seconds
2023-03-15 13:50:20.329 [INFO] 10.42.3.209: Loading extension module utils...
2023-03-15 13:50:20.329 [INFO] 10.42.3.209: Loading extension module utils...
2023-03-15 13:50:20.329 [INFO] 10.42.3.209: Loading extension module utils...
2023-03-15 13:50:20.330 [INFO] 10.42.3.209: Loading extension module utils...
2023-03-15 13:50:20.330 [INFO] 10.42.3.209: Loading extension module utils...
2023-03-15 13:50:20.330 [INFO] 10.42.3.209: Time to load utils op: 13.066633701324463 seconds
2023-03-15 13:50:20.330 [INFO] 10.42.3.209: Time to load utils op: 13.033191204071045 seconds
2023-03-15 13:50:20.330 [INFO] 10.42.3.209: Time to load utils op: 13.033248901367188 seconds
2023-03-15 13:50:20.330 [INFO] 10.42.3.209: Time to load utils op: 13.026963710784912 seconds
2023-03-15 13:50:20.330 [INFO] 10.42.3.209: Time to load utils op: 13.033659219741821 seconds
2023-03-15 13:50:20.370 [INFO] 10.42.3.209: Loading extension module utils...
2023-03-15 13:50:20.370 [INFO] 10.42.3.209: Time to load utils op: 13.025224208831787 seconds
2023-03-15 13:50:20.375 [INFO] 10.42.3.209: Loading extension module utils...
2023-03-15 13:50:20.375 [INFO] 10.42.3.209: Time to load utils op: 13.025392532348633 seconds
```

流水并行训练过程

```
[...]
[0.9, 0.999]]
2023-03-15 13:52:02.966 [INFO] 10.42.14.130: steps: 200 loss: 1.9796 iter time (s): 0.111 samples/sec: 287.830
2023-03-15 13:52:04.061 [INFO] 10.42.14.130: [2023-03-15 13:52:04, 058] [INFO] [logging.py:75:log_dist] [Rank 0] step=210, skipped=0, lr=[0.001], mom=[[0
.9, 0.999]]
2023-03-15 13:52:04.103 [INFO] 10.42.14.130: steps: 210 loss: 2.2377 iter time (s): 0.113 samples/sec: 282.176
2023-03-15 13:52:05.180 [INFO] 10.42.14.130: [2023-03-15 13:52:05, 180] [INFO] [logging.py:75:log_dist] [Rank 0] step=220, skipped=0, lr=[0.001], mom=[[0
.9, 0.999]]
2023-03-15 13:52:05.270 [INFO] 10.42.14.130: steps: 220 loss: 1.8330 iter time (s): 0.112 samples/sec: 286.874
2023-03-15 13:52:06.311 [INFO] 10.42.14.130: [2023-03-15 13:52:06, 310] [INFO] [logging.py:75:log_dist] [Rank 0] step=230, skipped=0, lr=[0.001], mom=[[0
.9, 0.999]]
2023-03-15 13:52:06.372 [INFO] 10.42.14.130: steps: 230 loss: 2.0810 iter time (s): 0.109 samples/sec: 292.368
2023-03-15 13:52:07.459 [INFO] 10.42.14.130: [2023-03-15 13:52:07, 459] [INFO] [logging.py:75:log_dist] [Rank 0] step=240, skipped=0, lr=[0.001], mom=[[0
.9, 0.999]]
2023-03-15 13:52:07.514 [INFO] 10.42.14.130: steps: 240 loss: 1.9840 iter time (s): 0.114 samples/sec: 280.791
2023-03-15 13:52:08.561 [INFO] 10.42.14.130: [2023-03-15 13:52:08, 561] [INFO] [logging.py:75:log_dist] [Rank 0] step=250, skipped=0, lr=[0.001], mom=[[0
.9, 0.999]]
2023-03-15 13:52:08.656 [INFO] 10.42.14.130: steps: 250 loss: 1.8471 iter time (s): 0.113 samples/sec: 283.082
2023-03-15 13:52:09.753 [INFO] 10.42.14.130: [2023-03-15 13:52:09, 753] [INFO] [logging.py:75:log_dist] [Rank 0] step=260, skipped=0, lr=[0.001], mom=[[0
.9, 0.999]]
2023-03-15 13:52:09.810 [INFO] 10.42.14.130: steps: 260 loss: 1.8117 iter time (s): 0.111 samples/sec: 287.790
2023-03-15 13:52:10.888 [INFO] 10.42.14.130: [2023-03-15 13:52:10, 888] [INFO] [logging.py:75:log_dist] [Rank 0] step=270, skipped=0, lr=[0.001], mom=[[0
.9, 0.999]]
2023-03-15 13:52:10.928 [INFO] 10.42.14.130: steps: 270 loss: 1.8820 iter time (s): 0.106 samples/sec: 302.517
2023-03-15 13:52:12.023 [INFO] 10.42.14.130: [2023-03-15 13:52:12, 023] [INFO] [logging.py:75:log_dist] [Rank 0] step=280, skipped=0, lr=[0.001], mom=[[0
.9, 0.999]]
2023-03-15 13:52:12.064 [INFO] 10.42.14.130: steps: 280 loss: 2.0265 iter time (s): 0.112 samples/sec: 285.573
```

DeepSpeed 使用

修改.py文件

1. DeepSpeed 使用 `argparse` library 为 DeepSpeed 运行时提供命令行配置。 使用

`deepspeed.add_config_arguments()` 将 DeepSpeed 的内置参数添加到应用程序的解析器中。

```
1 parser = argparse.ArgumentParser(description='CIFAR')
2     #data
3     # cuda
4 # train
5 parser.add_argument('--batch_size',
6                     default=32,
7                     type=int,
8                     help='mini-batch size (default: 32)')
9
10 parser.add_argument('--epochs',
11                     default=30,
12                     type=int,
13                     help='number of total epochs (default: 30)')
14
15 parser = deepspeed.add_config_arguments(parser)
16 args = parser.parse_args()
```

DeepSpeed 使用

修改.py文件

2. 使用 DeepSpeed 进行所有训练的入口点 `deepspeed.initialize()`.

返回值

Returns

A tuple of `engine`, `optimizer`, `training_dataloader`, `lr_scheduler`

`engine` : DeepSpeed runtime engine which wraps the client model for distributed training.

`optimizer` : Wrapped optimizer if a user defined `optimizer` is supplied, or if optimizer is specified in json config else `None`.

`training_dataloader` : DeepSpeed dataloader if `training_data` was supplied, otherwise `None`.

`lr_scheduler` : Wrapped lr scheduler if user `lr_scheduler` is passed, or if `lr_scheduler` specified in JSON configuration. Otherwise `None`.

1 例：

```
2 model_engine, optimizer, trainloader, __ = deepspeed.initialize(  
3     args=args, model=net, model_parameters=parameters, training_data=trainset)
```

DeepSpeed 使用

修改.py文件

3. 训练过程的数据、前向、后向都需要用model_engine

```
for data_iter_step, (samples, _) in enumerate(metric_logger.log_every(data_loader, print_freq, header)):
    # we use a per iteration (instead of per epoch) lr scheduler
    if data_iter_step % accum_iter == 0:
        lr_sched.adjust_learning_rate(optimizer, data_iter_step / len(data_loader) + epoch, args)
    samples = samples.to(device, non_blocking=True)
    #print("model_engine.local_rank=====", model_engine.local_rank)
    samples = samples.to(model_engine.local_rank, non_blocking=True)
    loss, _, _ = model_engine(samples)
    loss_value = loss.item()
    if not math.isfinite(loss_value):
        print("Loss is {}, stopping training".format(loss_value))
        sys.exit(1)
    loss /= accum_iter
    #loss_scaler(loss, optimizer, parameters=model_engine.parameters(),
    #            update_grad=(data_iter_step + 1) % accum_iter == 0)
    if (data_iter_step + 1) % accum_iter == 0:
        optimizer.zero_grad()
        torch.cuda.synchronize()
    #print("backward start=====loss=====", loss)
    model_engine.backward(loss, retain_graph=True)
    #print("backward end=====")
    model_engine.step()
```

DeepSpeed 使用

4. 修改 config.json 文件

```
"zero_optimization": {  
    "stage": [0|1|2|3],  
    "allgather_partitions": [true|false],  
    "allgather_bucket_size": 5e8,  
    "overlap_comm": false,  
    "reduce_scatter": [true|false],  
    "reduce_bucket_size": 5e8,  
    "contiguous_gradients" : [true|false],  
    "offload_param": {  
        ...  
    },  
    "offload_optimizer": {  
        ...  
    },  
    "stage3_max_live_parameters" : 1e9,  
    "stage3_max_reuse_distance" : 1e9,  
    "stage3_prefetch_bucket_size" : 5e8,  
    "stage3_param_persistence_threshold" : 1e6,  
    "sub_group_size" : 1e12,  
    "elastic_checkpoint" : [true|false],  
    "stage3_gather_16bit_weights_on_model_save": [true|false],  
    "ignore_unused_parameters": [true|false]  
    "round_robin_gradients": [true|false]  
}
```

DeepSpeed 使用

4. 修改 config.json 文件

```
"offload_optimizer": {  
    "device": "[cpu|nvme]",  
    "nvme_path": "/local_nvme",  
    "pin_memory": [true|false],  
    "buffer_count": 4,  
    "fast_init": false  
}  
  
"bf16": {  
    "enable": true  
}
```

参考文档:

<https://www.deepspeed.ai/docs/config-json/#zero-optimizations-for-fp16-training>

任务停止和查看log

1. 在分布式jupyter client中，新建notebook，选择pytorch1.5dl kernel
2. 查看log: dc.dl.log(unique_name= "torch-task_20210914105718"), 更多log /root/.dllog/...
3. 查看任务: dc.dl.task(name= " ")
4. 停止任务: dc.dl.stop(name= " ")

闪存使用-独享存储

目前分布式训练和单机训练使用同一个分布式存储池，由于分布式训练数据量和计算量较大，卡利用率长时间满负荷运行，存储的压力太大，因此独享闪存。

使用方法：

1. 准备执行文件：

```
cat cpdata.py
```

```
import os
cmd="cp -rf /root/share/glint360k_mask /root/share/fastcv/my"
ret=os.system(cmd)
```

```
print(ret)
print("Done!")
```

可替换自己所需的cmd

闪存使用-独享存储

2. 创建单个cpu worker资源， 默认规格：2c16g

```
export DC_DL_CONFIG='{"name":"task","framework":"pytorch","worker": {"count":1,"gpus":0}}'
```

3.启动单卡CPU任务，开始迁移数据

```
python dc_dl_run_nohup.py --name=cpdata --file=cpdata.py --queue_time=10 --nohup=true
```

4. 修改代码中数据文件的目录

```
dir_path= "/root/share/fastcv/my"
```

闪存使用-平台闪存

向文件申请flash额度

/root/flash



中国移动
China Mobile

谢谢

www.10086.cn