GPU 云服务器

AI 优化

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

AI 优化
训练加速引擎 TACO Train
概述
部署及实践

AI 优化
训练加速引擎 TACO Train
概述

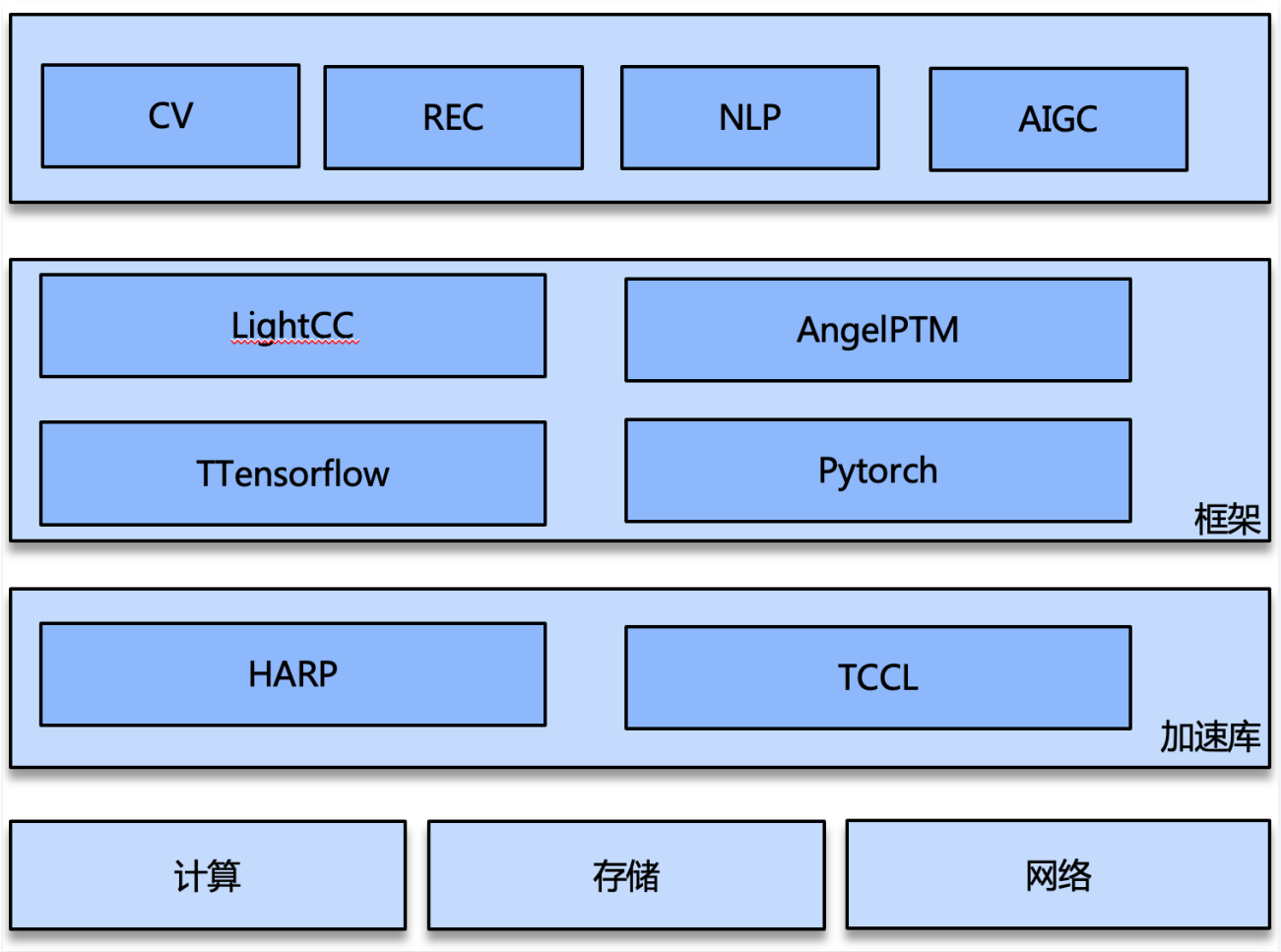
最近更新时间：2024-11-14 11:27:42

背景信息

近几年随着 AI 模型参数的倍增及训练数据的日益增长，用户对模型迭代效率的需求也随之增长，单个 GPU 的算力和显存资源已无法满足大部分业务场景，使用单机多卡或多机多卡训练已成为趋势。单机多卡训练场景的参数同步借助 NVIDIA NVLINK 技术，基本可以获得较高的线性扩展比，但多机多卡训练场景严重依赖多机之间的网络互联技术。网卡厂商提供了高速互联技术 Infiniband 或 RoCE，虽使多机通信效率大幅提升，但成本也大幅增加。如何在普通甚至低速网络环境下提升分布式系统的训练效率，已成为用户关注的焦点。

TACO Train 简介

目前业内已有一些成熟的分布式训练加速技术，例如多级通信、多流通信、梯度融合、压缩通信等，TACO Train 也引入了类似的加速技术。同时，TACO Train 推出了自定义的用户态协议栈 HARP，有效解决普通网络环境下的多机网络通信问题。
TACO Train 是腾讯云**异构计算团队**基于 IaaS 资源推出的 AI 训练加速引擎，为用户提供开箱即用的 AI 训练套件。TACO Train基于腾讯内部丰富的 AI 业务场景，提供自底向上的网络通信、分布式策略及训练框架等多层级的优化，是一套全生态的训练加速方案。



目前TACO Train 提供了5个训练加速组件：

- **AngelPTM**：大模型预训练框架，支持 GPT/T5/BERT 等大模型。
- **TCCL**：针对腾讯云星脉网络架构的高性能定制加速通信库，为 AI 大模型训练提供更高效的网络通信性能。
- **HARP**：自研用户态网络协议栈，加速普通网络环境的多机通信效率。
- **LightCC**：分布式训练框架，支持 Horovod 或者 pytorch DDP 的分布式训练加速。
- **Tencent Tensorflow 1.15**：基于 Tensorflow 1.15深度优化的训练框架（以下简称 TTF），支持搜索广告推荐场景下海量数据特征的增量训练。

AngelPTM

AngelPTM 是基于 DeepSpeed 和 Megatron 深度定制开发的大模型训练框架，支持 NLP/多模态/AIGC 等多类预训练任务。由于大模型的参数规模巨大，对硬件存储资源提出了挑战，AngelPTM 支持以更少的资源和更快的速度训练大模型，兼容社区方案 API ，支持业务快速接入，

- ZeRO Cache 策略：基于 ZeRO 策略把内存作为二级存储 offload 参数、梯度、优化器状态到 CPU 内存（也支持 SSD 作为第三级存储）。ZeRO-Cache 为了最大化利用内存和显存进行模型状态的缓存，引入了显存内存统一存储视角，将存储容量的上界由内存扩容到内存+显存总和。同时将多流异步化做到了高效，在 GPU 计算的同时进行数据 IO 和 NCCL 通信，使用异构流水线均衡设备间的负载，最大化提升整个系统的吞吐。ZeRO-Cache 将 GPU 显存、CPU 内存统一视角管理，击破了异构存储的壁垒，减少了冗余存储和内存碎片，增加了内存的利用率，极大扩充了模型存储可用空间。

- 自动流水并行：3D 并行是 Megatron 的原始能力，但是 Megatron 的流水并行只支持 block 级别（Transformer Layer），且需要依靠专家经验人工指定 stage 切分以确保 stage 间负载均衡；自动流水并行可以做到 op 级别，且自动 profile op 性能，自动搜索负载均衡的切分方案后执行流水并行训练。
- 高性能 MoE 组件：基于拓扑感知的 AlltoAll 通信加速专家并行，提供高性能定制算子，支持计算通信流水提高迭代效率。

TCCL

TCCL（Tencent Collective Communication Library）是一款针对腾讯云星脉网络架构的高性能定制加速通信库。主要功能是依托星脉网络硬件架构，为 AI 大模型训练提供更高效的网络通信性能，同时具备网络故障快速感知与自愈的智能运维能力。TCCL 基于开源的 NCCL 代码做了扩展优化，完全兼容 NCCL 的功能与使用方法。TCCL 目前支持主要特性包括：

- 双网口动态聚合优化，发挥 bonding 设备的性能极限。
- 全局 Hash 路由（Global Hash Routing），负载均衡，避免拥塞。
- 拓扑亲和性流量调度，最小化流量绕行。

HARP

随着网络硬件技术的发展，网络带宽从 10Gbps 增长到 100Gbps 甚至更高，在数据中心大量部署使用。但目前普遍使用的内核网络协议栈存在着一些必要的开销，使其不能很好地利用高速网络设备。为解决该问题，腾讯云自研了用户态网络协议栈 HARP，可以以 Plug-in 的方式集成到 NCCL 中，无需任何业务改动，加速云上分布式训练性能。在 VPC 的环境下，相比传统的内核协议栈，HARP 提供了以下的能力：

- 支持全链路内存零拷贝，HARP 协议栈提供特定的 buffer 给应用，使应用的数据经过 HARP 协议栈处理后由网卡直接进行收发，消除内核协议栈中耗时及占用 CPU 较高的多次内存拷贝操作。
- 支持协议栈多实例隔离，即应用可以在多个 CPU core 上创建特定协议栈实例处理网络报文，每个实例间相互隔离，保证性能线性增长。
- 数据平面无锁设计，HARP 协议栈内部保证网络 session 的数据仅在创建该 session 的 CPU core 上，使用特定的协议栈实例处理。减少了内核中同步锁的开销，也降低了 CPU 的 Cache Miss 率，大幅提升网络数据的处理性能。

LightCC

LightCC 对社区分布式方案的通信策略进行了深度定制优化，完全兼容 Horovod 或者 pytorch DDP API，支持业务快速接入。主要包括的优化能力如下：

- 2D AllReduce 充分利用通信带宽。
- 高效的梯度融合方式。
- TOPK/FP16 压缩通信，降低通信量，提高传输效率。

TTF

TensorFlow 是深度学习领域中应用最广泛的开源框架之一，但在很多业务场景下，开源 Tensorflow 有其特定的限制。为了解决实际业务中遇到的问题，Tencent Tensorflow（以下简称 TTF）提供了以下能力：

- 相比原始的静态 Embedding，高维稀疏动态 Embedding 帮助用户在不需要重新训练的条件下，动态添加和删除特征，按需使用内存，避免 Hash 冲突，同时保留原始 TF 的 API 设计风格。
- 混合精度在原有实现的基础上增加了调整精度的策略，根据 loss 的状态自动在全精度和半精度之间切换，避免精度损失。
- 针对特定业务场景的 XLA，Grappler 图优化，以及自适应编译框架解决冗余编译的问题。
- 开源 TF 1版本不再提供对 Ampere GPU 的支持，但考虑到较多用户仍在使用 TF 1.15版本的问题，TTF 添加了对 CUDA 11的支持，让用户可以使用 A100来进行模型训练。

部署及实践

最近更新时间：2024-08-23 14:54:11

操作场景

本文介绍如何基于云服务器 CVM 搭建 torch+Taco Train 分布式训练集群，更多最佳实践请参见 计算加速套件 TACO Kit 文档 。

操作步骤

购买实例

购买实例，其中实例、存储及镜像请参考以下信息选择，其余配置请参见 通过购买页创建实例 按需选择。

- **实例**：选择 计算型 GN10Xp 或 计算型 GT4 。
- **系统盘**：配置容量不小于50GB的云硬盘。您也可在创建实例后使用文件存储，详情参见 在 Linux 客户端上使用 CFS 文件系统 。
- **镜像**：建议选择**公共镜像**，支持自动安装 GPU 驱动。若您选择**自定义镜像**，则需要自行安装 GPU 驱动。
- 操作系统请使用 CentOS 8.0、CentOS 7.8、Ubuntu 20.04、Ubuntu 18.04、TencentOS 3.1、TencentOS 2.4。
- 若您选择**公共镜像**，则请勾选**后台自动安装 GPU 驱动**，实例将在系统启动后预装对应版本驱动。如下图所示：

配置实例环境

验证 GPU 驱动

1. 参见 使用标准登录方式登录 Linux 实例 ，登录实例。
2. 执行以下命令，验证 GPU 驱动是否安装成功。

```
nvidia-smi
```

查看输出结果是否为 GPU 状态：

- 是，代表 GPU 驱动安装成功。
- 否，请参见 NVIDIA Driver Installation Quickstart Guide 进行安装。

配置 HARP 分布式训练环境

1. 参见 配置 HARP 分布式训练环境，配置所需环境。
2. 配置完成后，执行以下命令进行验证，若配置文件存在，则表示已配置成功。

```
ls /usr/local/tfabric/tools/config/ztcp*.conf
```

安装 docker 和 nvidia docker

1. 执行以下命令，安装 docker。

```
curl -s -L http://mirrors.tencent.com/install/GPU/taco/get-docker.sh |
sudo bash
```

若您无法通过该命令安装，请尝试多次执行命令，或参见 Docker 官方文档 Install Docker Engine 进行安装。
本文以 CentOS 为例，安装成功后，返回结果如下图所示：

```
+ sh -c 'yum install -y -q docker-ce-rootless-extras'
Package docker-ce-rootless-extras-20.10.16-3.el7.x86_64 already installed and latest version

================================================================================

To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:

    dockerd-rootless-setuptool.sh install

Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.


To run the Docker daemon as a fully privileged service, but granting non-root
users access, refer to https://docs.docker.com/go/daemon-access/

WARNING: Access to the remote API on a privileged Docker daemon is equivalent
         to root access on the host. Refer to the 'Docker daemon attack surface'
         documentation for details: https://docs.docker.com/go/attack-surface/

================================================================================
```

2. 执行以下命令，安装 nvidia-docker2。

```
curl -s -L http://mirrors.tencent.com/install/GPU/taco/get-nvidia-docker2.sh | sudo bash
```

若您无法通过该命令安装，请尝试多次执行命令，或参见 NVIDIA 官方文档 Installation Guide & mdash 进行安装。

本文以 CentOS 为例，安装成功后，返回结果如下图所示：

```
Running transaction
  Installing : libnvidia-container1-1.9.0-1.x86_64
  Installing : libnvidia-container-tools-1.9.0-1.x86_64
  Installing : nvidia-container-toolkit-1.9.0-1.x86_64
  Installing : nvidia-docker2-2.10.0-1.noarch
  Verifying  : libnvidia-container-tools-1.9.0-1.x86_64
  Verifying  : nvidia-container-toolkit-1.9.0-1.x86_64
  Verifying  : nvidia-docker2-2.10.0-1.noarch
  Verifying  : libnvidia-container1-1.9.0-1.x86_64

Installed:
  nvidia-docker2.noarch 0:2.10.0-1

Dependency Installed:
  libnvidia-container-tools.x86_64 0:1.9.0-1       libnvidia-container1.x86_64 0:1.9.0-1       nvidia-container-toolkit.x86_64 0:1.9.0-1

Complete!
```

下载 docker 镜像

执行以下命令，下载 docker 镜像。

```
docker pull ccr.ccs.tencentyun.com/qcloud/taco-train:torch111-cu113-cvm-0.4.3
```

该镜像包含的软件版本信息如下：

- OS：Ubuntu 20.04.4 LTS
- python：3.8.10
- cuda toolkits：V11.3.109
- cudnn library：8.2.0
- nccl library：2.9.9
- **tencent-lightcc：3.1.1**
- **HARP library：v1.3**
- **torch：1.11.0+cu113**

其中：

- LightCC 是腾讯云提供的基于 Horovod 深度定制优化的通信组件，完全兼容 Horovod API，不需要任何业务适配。
- HARP 是腾讯云提供的用户态协议栈，致力于提高 VPC 网络下的分布式训练的通信效率。以动态库的形式提供，官方 NCCL 初始化过程中会自动加载，不需要任何业务适配。
- torch 是官方版本1.11.0版本。

启动 docker 镜像

执行以下命令，启动 docker 镜像。

```
docker run -it --rm --gpus all --privileged --net=host -v /sys:/sys -v /dev/hugepages:/dev/hugepages -v /usr/local/tfabric/tools:/usr/local/tfabric/tools ccr.ccs.tencentyun.com/qcloud/taco-train:torch111-cu113-cvm-0.4.3
```

> **注意**
> `/dev/hugepages` 和 `/usr/local/tfabric/tools` 包含了 HARP 运行所需要的大页内存和配置文件。

分布式训练 benchmark 测试

> **说明**

> docker 镜像中的文件 `/mnt/tensorflow_synthetic_benchmark.py` 来自 horovod example 。

单卡

执行以下命令，进行测试。

```
/usr/local/openmpi/bin/mpirun -np 1 --allow-run-as-root -bind-to
none -map-by slot -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=eth0 -
x LD_LIBRARY_PATH -x PATH -mca btl_tcp_if_include eth0 python3
/mnt/pytorch_synthetic_benchmark.py --model=vgg16 --batch-
size=128
```

下图为 GN10Xp/V100的单卡 benchmark 结果：

```
Model: vgg16
Batch size: 128
Number of GPUs: 1
Running warmup...
Running benchmark...
Iter #0: 253.0 img/sec per GPU
Iter #1: 252.1 img/sec per GPU
Iter #2: 252.7 img/sec per GPU
Iter #3: 251.2 img/sec per GPU
Iter #4: 251.7 img/sec per GPU
Iter #5: 251.1 img/sec per GPU
Iter #6: 251.1 img/sec per GPU
Iter #7: 250.9 img/sec per GPU
Iter #8: 250.3 img/sec per GPU
Iter #9: 250.8 img/sec per GPU
Img/sec per GPU: 251.5 +-1.6
Total img/sec on 1 GPU(s): 251.5 +-1.6
```

单机多卡

执行以下命令，进行测试。

```
/usr/local/openmpi/bin/mpirun -np 8 --allow-run-as-root -bind-to
none -map-by slot -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=eth0 -
x LD_LIBRARY_PATH -x PATH -mca btl_tcp_if_include eth0 python3
```

```
/mnt/pytorch_synthetic_benchmark.py --model=vgg16 --batch-
size=128
```

下图为 GN10Xp/V100的单机8卡 benchmark 结果：

```
Model: vgg16
Batch size: 128
Number of GPUs: 8
Running warmup...
Running benchmark...
Iter #0: 248.0 img/sec per GPU
Iter #1: 248.0 img/sec per GPU
Iter #2: 247.3 img/sec per GPU
Iter #3: 247.3 img/sec per GPU
Iter #4: 247.5 img/sec per GPU
Iter #5: 247.0 img/sec per GPU
Iter #6: 246.9 img/sec per GPU
Iter #7: 246.7 img/sec per GPU
Iter #8: 246.9 img/sec per GPU
Iter #9: 246.3 img/sec per GPU
Img/sec per GPU: 247.2 +-1.0
Total img/sec on 8 GPU(s): 1977.4 +-8.2
```

多机多卡

1. 参见 购买实例 – 启动 docker 镜像 步骤，购买和配置多台训练机器。
2. 配置多台服务器 docker 间相互免密访问，详情请参见 配置容器 SSH 免密访问 。
3. 执行以下命令，使用 TACO Train 进行多机训练加速。

```
/usr/local/openmpi/bin/mpirun -np 16 -H gpu1:8,gpu2:8 --allow-
run-as-root -bind-to none -map-by slot -x NCCL_ALGO=RING -x
NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=eth0 -x
HOROVOD_MPI_THREADS_DISABLE=1 -x HOROVOD_FUSION_THRESHOLD=0  -x
HOROVOD_CYCLE_TIME=0 -x LIGHT_INTRA_SIZE=8 -x
LIGHT_2D_ALLREDUCE=1 -x LIGHT_TOPK_ALLREDUCE=1 -x
LIGHT_TOPK_THRESHOLD=2097152 -x LD_LIBRARY_PATH -x PATH -mca
btl_tcp_if_include eth0 python3
/mnt/pytorch_synthetic_benchmark.py --model=vgg16 --batch-
size=128
```

下图为 GN10Xp/V100的2机16卡 benchmark 结果：

```
Running benchmark...
Iter #0: 222.9 img/sec per GPU
Iter #1: 225.2 img/sec per GPU
Iter #2: 222.9 img/sec per GPU
Iter #3: 225.3 img/sec per GPU
Iter #4: 221.7 img/sec per GPU
Iter #5: 219.5 img/sec per GPU
Iter #6: 224.5 img/sec per GPU
Iter #7: 221.6 img/sec per GPU
Iter #8: 221.7 img/sec per GPU
Iter #9: 221.1 img/sec per GPU
Img/sec per GPU: 222.6 +-3.5
Total img/sec on 16 GPU(s): 3562.2 +-55.9
```

LightCC 的环境变量说明如下表：

环境变量	默认值	说明
LIGHT_2D_ALLREDUCE	0	是否使用 2D-Allreduce 算法
LIGHT_INTRA_SIZE	8	2D-Allreduce 组内 GPU 数
LIGHT_HIERARCHICAL_THRESHOLD	1073741824	2D-Allreduce 的阈值，单位是字节，小于等于该阈值的数据才使用 2D-Allreduce
LIGHT_TOPK_ALLREDUCE	0	是否使用 TOPK 压缩通信
LIGHT_TOPK_RATIO	0.01	使用 TOPK 压缩的比例
LIGHT_TOPK_THRESHOLD	1048576	TOPK 压缩的阈值，单位是字节，大于等于该阈值的数据才使用 TOPK 压缩通信
LIGHT_TOPK_FP16	0	压缩通信的 value 是否转成 FP16

4. 执行以下命令，关闭 TACO LightCC 加速进行测试。

```
# 修改环境变量，使用Horovod进行多机Allreduce
/usr/local/openmpi/bin/mpirun -np 16 -H gpu1:8,gpu2:8 --allow-run-as-root -bind-to none -map-by slot -x NCCL_ALGO=RING -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=eth0 -x LD_LIBRARY_PATH -x PATH -mca btl_tcp_if_include eth0 python3 /mnt/pytorch_synthetic_benchmark.py --model=vgg16 --batch-size=128
```

下图为 GN10Xp/V100的2机16卡，关闭 LightCC 后的 benchmark 结果：

```
Model: vgg16
Batch size: 128
Number of GPUs: 16
Running warmup...
Running benchmark...
Iter #0: 103.4 img/sec per GPU
Iter #1: 103.4 img/sec per GPU
Iter #2: 103.7 img/sec per GPU
Iter #3: 103.2 img/sec per GPU
Iter #4: 103.8 img/sec per GPU
Iter #5: 103.6 img/sec per GPU
Iter #6: 104.0 img/sec per GPU
Iter #7: 103.2 img/sec per GPU
Iter #8: 103.4 img/sec per GPU
Iter #9: 95.8 img/sec per GPU
Img/sec per GPU: 102.8 +-4.6
Total img/sec on 16 GPU(s): 1644.1 +-73.5
```

5. 执行以下命令，同时关闭 LightCC 和 HARP 加速进行测试。

```
# 将HARP加速库rename为bak.libnccl-net.so即可关闭HARP加速。/usr/local/openmpi/bin/mpirun -np 2 -H gpu1:1,gpu2:1 --allow-run-as-root -bind-to none -map-by slot mv /usr/lib/x86_64-linux-gnu/libnccl-net.so /usr/lib/x86_64-linux-gnu/bak.libnccl-net.so# 修改环境变量，使用Horovod进行多机Allreduce
/usr/local/openmpi/bin/mpirun -np 16 -H gpu1:8,gpu2:8 --allow-run-as-root -bind-to none -map-by slot -x NCCL_ALGO=RING -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=eth0 -x LD_LIBRARY_PATH -x PATH -mca btl_tcp_if_include eth0 python3 /mnt/pytorch_synthetic_benchmark.py --model=vgg16 --batch-size=128
```

下图为 GN10Xp/V100的2机16卡，同时关闭 LightCC 和 HARP 后的 benchmark 结果：

```
Model: vgg16
Batch size: 128
Number of GPUs: 16
Running warmup...
Running benchmark...
Iter #0: 52.8 img/sec per GPU
Iter #1: 51.4 img/sec per GPU
Iter #2: 51.6 img/sec per GPU
Iter #3: 53.7 img/sec per GPU
Iter #4: 54.1 img/sec per GPU
Iter #5: 54.0 img/sec per GPU
Iter #6: 53.0 img/sec per GPU
Iter #7: 54.1 img/sec per GPU
Iter #8: 52.9 img/sec per GPU
Iter #9: 52.2 img/sec per GPU
Img/sec per GPU: 53.0 +-1.9
Total img/sec on 16 GPU(s): 847.8 +-30.1
```

注意：
测试完如需恢复 HARP 加速能力，只需要把所有机器上的 bak.libnccl-net.so 重新命名为 libnccl-net.so 即可。

总结

本文测试数据如下：

机器：GN10Xp（V100 * 8）+ 25G VPC
容器：ccr.ccs.tencentyun.com/qcloud/taco-train:torch111-cu113-cvm-0.4.1
网络模型：VGG16Batch：128
数据：synthetic data

机型	GPU 卡数	Horovod+TCP		Horovod+HARP		LightCC+HARP	
		性能（img/sec）	线性加速比	性能（img/sec）	线性加速比	性能（img/sec）	线性加速比
GN10Xp/V100	1	251	-	251	-	251	-
	8	1977	98%	1977	98%	1977	98%
	16	847	21%	1644	40%	3562	88%

说明如下：

- 对于 GN10Xp，相比开源方案，使用 TACO 分布式训练加速组件之后，16卡V100的线性加速比从93%提升到97%（由于当前网络模型的加速已经很高，软件优化提升的空间有限）。
- LightCC 和 HARP 只在多机分布式训练当中才有加速效果，单机8卡场景由于 NVLink 的高速带宽存在，一般不需要额外的加速就能达到比较高的线性加速比。
- 上述 benchmark 脚本也可以支持除了 VGG16之外的其他模型。ModelName 请参见 Keras Applications 。
- 上述 docker 镜像仅用于 demo，若您具备开发或者部署环境，请提供 OS/python/CUDA/torch 版本信息，并联系腾讯云售后提供特定版本的 TACO 加速组件。