曦云®系列通用计算 GPU

mcPyTorch2.0 用户指南

CSOG-23031-020-F3_V05　　2024-09-06

沐曦专有和三级保密信息　　本文档受 NDA 管控

更新记录

版本	日期	更新说明
V05	2024-09-06	更新以下章节： 1 概述 2 快速安装 3.2.3 Dropout Layers 3.11.2 torch.backends.cudnn 新增以下章节： 4 常用环境变量
V04	2024-07-30	更新以下章节： 1 概述 2.1.1 环境准备 2.1.2 开始安装 3 功能支持
V03	2024-03-22	新增曦云®系列 GPU 产品信息 新增以下章节： 2.2 使用 Docker 运行 5 精度对比工具
V02	2023-12-29	描述性修改和格式修正 更新以下章节： 2.1.1 环境准备 2.1.2 开始安装
V01	2023-10-16	正式版本首次发布

目录

图目录

1. 概述

mcPyTorch 在 PyTorch 的基础上增加了 MXMACA®后端，支持使用 MXMACA 硬件加速 PyTorch 中各类计算任务。

当前发布包含以下版本安装包：

- PyTorch 2.0.0 + Python 3.8
- PyTorch 2.0.0 + Python 3.10
- PyTorch 2.1.2 + Python 3.8
- PyTorch 2.1.2 + Python 3.10

支持 x86_64 架构下 Ubuntu 18/20/22 以及 CentOS 7/8/9 系统上运行。mcPyTorch 的具体支持功能，参见 3 功能支持。

2. 快速安装

2.1 基于 pip 安装

2.1.1 环境准备

- Python

  匹配目标安装包的 Python 版本（例如，Python 3.8 或者 Python 3.10）

- MXMACA 环境

  安装 Driver 软件包以及 MXMACA SDK 软件包

- cu-bridge 环境

  - 在安装 MXMACA 环境到 **/opt/maca/** 目录后，将 cu-bridge 解压到 **/opt/maca/tools/** 目录下，具体操作步骤参见相关文档。可以检验 **/opt/maca/tools/cu-bridge/bin/cucc** 文件是否存在，从而判断 cu-bridge 环境是否配置完成。
  - cu-bridge 环境不需要安装 CUDA Toolkit。
  - 如果不使用 PyTorch Extension 功能可以不配置 cu-bridge 环境。PyTorch Extension 功能，参见官网描述。

2.1.2 开始安装

PyTorch 安装包以 **maca-pytorch${pytorch_version}-py${python_version}-${MACA_version}-${arch_info}.tar.xz** 格式命名。例如，PyTorch 2.1 + Python 3.10 版本的安装包名称为：**maca-pytorch2.1-py310-2.24.0.1-x86_64.tar.xz**。

获取 PyTorch 安装包后，解压可以得到包括 mcPyTorch、mcTorchvison、mcTorchaudio、mcApex、mcFlashattn 在内的 whl 格式安装包。PyTorch 2.1 版本的安装包中提供了 mcTriton 的安装包。

操作步骤

安装方式同标准 whl 包。

1. 安装 mcPyTorch：

   ```
   python -m pip install torch-*.whl
   ```

2. 安装 mcFlashattn：

   ```
   python -m pip install flash_attn-*.whl
   ```

3. （可选）安装 mcTorchvision：

```
python -m pip install torchvision-*.whl
```

4. （可选）安装 mcTorchaudio：

```
python -m pip install torchaudio-*.whl
```

5. （可选）安装 mcApex：

```
python -m pip install apex-*.whl
```

6. （可选）安装 mcTriton：

```
python -m pip install triton-*.whl
```

2.1.3 验证安装

操作步骤

1. 运行前设置环境变量：

```
export MACA_PATH=/opt/maca/
export
LD_LIBRARY_PATH=${MACA_PATH}/lib:${MACA_PATH}/mxgpu_llvm/lib:${LD_LIBRARY
_PATH}
export MACA_CLANG_PATH=${MACA_PATH}/mxgpu_llvm/bin
export PYTORCH_USE_FLASHATTN=1
```

如果使用 PyTorch Extension 功能，需要安装 cu-bridge 环境，并额外设置以下环境变量：

```
export CUDA_PATH=/opt/maca/tools/cu-bridge/
export CUCC_PATH=/opt/maca/tools/cu-bridge/
export PATH=${CUCC_PATH}/tools:${CUCC_PATH}/bin:${PATH}
```

2. 执行以下命令：

```
python -c "import torch; print(torch.ones(2).cuda())"
```

会得到如下打印输出，即表明安装成功：

```
tensor([1., 1.], device='cuda:0')
```

2.1.4 如何卸载

卸载方式同标准 whl 包。

操作步骤

1. 卸载 mcPyTorch：

```
python -m pip uninstall torch
```

2.2 使用 Docker 运行

MXMACA 容器镜像是以离线形式发布。用户可在随本文档发布的软件包中找到相关压缩包。本文档中以 **mxc500-torch2.0-py38-mc2.23.0.6-ubuntu20.04-x86_64.container.xz** 为例，用户应根据实际收到的软件包版本对版本字段进行相应替换。

操作步骤

1. 获取 MXMACA 容器镜像，执行以下命令，完成容器镜像的加载：

```
docker load < ./mxc500-torch2.0-py38-mc2.23.0.6-ubuntu20.04-x86_64.container.xz
```

2. 在 Docker 容器中使用板卡，使用全部曦云系列 GPU（以 C500 为例）：

```
docker run -it --device=/dev/mxcd --device=/dev/dri --group-add video mxc500-torch2.0-py38:mc2.23.0.6-ubuntu20.04-x86_64 /bin/bash
```

3. 功能支持

mcPyTorch 以兼容 PyTorch 的原生使用方式为设计目标。大部分情况下，用户可以参考官方文档获得 mcPyTorch 的使用方式。

本章参照 PyTorch 官方文档内容，介绍 mcPyTorch 各模块使用的兼容性和注意事项。其中：

- 支持：支持并兼容原生用法
- 部分支持：部分支持原生用法，或 API 行为和原生用法有区别

3.1 torch

支持

3.2 torch.nn

支持

3.2.1 Convolution Layers

支持：

- nn.Conv1d
- nn.Conv2d
- nn.Conv3d
- nn.ConvTranspose1d
- nn.ConvTranspose2d
- nn.ConvTranspose3d
- nn.LazyConv1d
- nn.LazyConv2d
- nn.LazyConv3d
- nn.LazyConvTranspose1d
- nn.LazyConvTranspose2d
- nn.LazyConvTranspose3d

- nn.Unfold
- nn.Fold

说明

默认情况下，conv 的 FP32 数据类型没有使用 TF32 而使用 FP32 进行计算加速，可以使用 `torch.backends.cudnn.allow_tf32=True` 打开。

3.2.2 Recurrent Layers

支持：

- nn.RNNBase
- nn.RNN
- nn.LSTM
- nn.GRU
- nn.RNNCell
- nn.LSTMCell
- nn.GRUCell

3.2.3 Dropout Layers

支持：

- nn.Dropout
- nn.Dropout1d
- nn.Dropout2d
- nn.Dropout3d
- nn.AlphaDropout
- nn.FeatureAlphaDropout

说明

PyTorch 中 Dropout 类运算随机行为和硬件参数相关，因此 MXMACA 设备和 CUDA 设备上默认运行结果随机行为不一致。

3.3 torch.nn.functional

支持

3.4 torch.Tensor

支持

3.5 Tensor Attributes

支持

3.6 Tensor Views

支持：

- basic slicing and indexing
- adjoint
- as_strided
- detach
- diagonal
- expand
- expand_as
- movedim
- narrow
- permute
- select
- squeeze
- transpose
- t
- T
- H
- mT

- mH
- real
- imag
- view_as_real
- unflatten
- unfold
- unsqueeze
- view
- view_as
- unbind
- split
- hsplit
- vsplit
- tensor_split
- split_with_sizes
- swapaxes
- swapdims
- chunk
- indices (sparse tensor only)
- values (sparse tensor only)

3.7 torch.amp

支持

3.8 torch.autograd

支持

3.9 torch.library

支持

3.10 torch.cuda

3.10.1 cuda

支持：

- StreamContext
- can_device_access_peer
- current_blas_handle
- current_device
- current_stream
- default_stream
- device
- device_count
- device_of
- get_sync_debug_mode
- init
- ipc_collect
- is_available
- is_initialized
- set_device
- set_stream
- set_sync_debug_mode
- stream
- synchronize
- OutOfMemoryError

部分支持：

- get_arch_list

  返回 `['sm_80']`。模拟 `CUDA capability==8.0` 时的行为。

- get_device_capability

  返回 `(8. 0)`。模拟 `CUDA capability==8.0` 时的行为。

- get_device_name

  返回 MXMACA 硬件相应设备名，而非 NVIDIA 设备名例如：NVIDIA A100-PCIE-40GB。

- get_device_properties

  返回对应 CUDA 架构描述的 MXMACA 架构信息。

- get_gencode_flags

  返回`"-gencode compute=compute_80,code=sm_80"`。模拟 `CUDA capability==8.0` 时的行为。

3.10.2 Random Number Generator

支持：

- get_rng_state
- get_rng_state_all
- set_rng_state
- set_rng_state_all
- manual_seed
- manual_seed_all
- seed
- seed_all
- initial_seed

3.10.3 Communication collectives

支持：

- comm.broadcast
- comm.broadcast_coalesced

- comm.reduce_add
- comm.scatter
- comm.gather

3.10.4 Streams and events

支持：

- Stream
- Event

3.10.5 Memory management

支持：

- empty_cache
- list_gpu_processes
- mem_get_info
- memory_stats
- memory_summary
- memory_snapshot
- memory_allocated
- max_memory_allocated
- reset_max_memory_allocated
- memory_reserved
- max_memory_reserved
- set_per_process_memory_fraction
- memory_cached
- max_memory_cached
- reset_max_memory_cached
- reset_peak_memory_stats
- caching_allocator_alloc
- caching_allocator_delete

- get_allocator_backend
- change_current_allocator

3.10.6 Tools Extension

支持：

- nvtx.mark
- nvtx.range_push
- nvtx.range_pop

3.11 torch.backends

3.11.1 torch.backends.cuda

支持：

- torch.backends.cuda.is_built

  返回 `True`，表示 mcPyTorch 包含 MXMACA backends。
- torch.backends.cuda.matmul.allow_tf32
- torch.backends.cuda.matmul.allow_fp16_reduced_precision_reduction
- torch.backends.cuda.matmul.allow_bf16_reduced_precision_reduction
- torch.backends.cuda.cufft_plan_cache
- torch.backends.cuda.math_sdp_enabled
- torch.backends.cuda.enable_math_sdp
- torch.backends.cuda.sdp_kernel(enable_flash=True, enable_math=True, enable_mem_efficient=True)

  目前仅支持 `enable_math=True`

3.11.2 torch.backends.cudnn

支持：

- torch.backends.cudnn.is_available
- torch.backends.cudnn.enabled

- torch.backends.cudnn.allow_tf32
- torch.backends.cudnn.deterministic
- torch.backends.cudnn.benchmark
- torch.backends.cudnn.benchmark_limit
- torch.backends.cudnn.version

说明

默认情况下，`torch.backends.cudnn.allow_tf32` 为 `False`，`torch.backends.cuda.matmul.allow_tf32` 为 `False`。

3.11.3 torch.backends.mps

支持：

- torch.backends.mps.is_available
- torch.backends.mps.is_built

说明

mcPyTorch 不包含 MPS backends，上述两个接口总是返回 `False`。

3.11.4 torch.backends.mkl

支持：

- torch.backends.mkl.is_available
- torch.backends.mkl.verbose

3.11.5 torch.backends.mkldnn

支持：

- torch.backends.mkldnn.is_available
- torch.backends.mkldnn.verbose

3.11.6 torch.backends.openmp

支持：

- torch.backends.openmp.is_available

3.11.7 torch.backends.opt_einsum

3.12 torch.distributed

支持

3.13 torch.distributions

支持

3.14 torch._dynamo

支持

3.15 torch.fft

支持

3.16 torch.fx

支持

3.17 torch.jit

支持

3.18 torch.linalg

支持

3.19 torch.package

支持

3.20 torch.profiler

支持

3.21 torch.nn.init

支持

3.22 torch.onnx

支持

3.23 torch.optim

支持

3.24 Complex Numbers

支持

3.25 DDP Communication Hooks

支持

3.26 torch.random

支持

3.27 torch.masked

支持

3.28 torch.nested

支持

3.29 torch.sparse

支持

3.30 torch.Storage

支持

3.31 torch.testing

支持

3.32 torch.utils.benchmark

支持

3.33 torch.utils.bottleneck

支持

3.34 torch.utils.checkpoint

支持

3.35 torch.utils.cpp_extension

支持

说明

Extension 的编译需要 cu-bridge 的支持，请参考相关文档配置 cu-bridge 环境。

3.36 torch.utils.data

支持

3.37 torch.utils.jit

支持

3.38 torch.utils.model_zoo

支持

3.39 torch.utils.tensorboard

支持

3.40 Type Info

支持

3.41 torch.__config__

支持

4. 常用环境变量

PYTORCH_DEFAULT_NCHW

对于四维 Tensor，mcPyTorch 在其从 CPU 向 device 侧搬运时会更改 layout 为 torch.channels_last (NHWC)。该环境变量用于恢复官方默认行为，即搬运时不做额外的 layout 改变逻辑。

- 设置环境变量：`export PYTORCH_DEFAULT_NCHW=1`
- 取消设置：`unset PYTORCH_DEFAULT_NCHW`

示例：

- 执行 `unset PYTORCH_DEFAULT_NCHW`，显示如下信息：

```
>>> import torch
>>> a = torch.rand(2,3,4,5).cuda()
>>> a.shape
torch.Size([2, 3, 4, 5])
>>> a.stride()
(60, 1, 15, 3)
```

- 执行 `export PYTORCH_DEFAULT_NCHW=1`，显示如下信息：

```
>>> import torch
>>> a = torch.rand(2,3,4,5).cuda()
>>> a.shape
torch.Size([2, 3, 4, 5])
>>> a.stride()
(60, 20, 5, 1)
```

TORCH_ALLOW_TF32_CUBLAS_OVERRIDE

`torch.backends.cuda.matmul.allow_tf32` 默认为 `False`。设置该环境变量时控制 `torch.backends.cuda.matmul.allow_tf32` 默认为 `True`。

- 设置：`export TORCH_ALLOW_TF32_CUBLAS_OVERRIDE =1`
- 取消设置：`unset TORCH_ALLOW_TF32_CUBLAS_OVERRIDE`

PYTORCH_ALLOW_CUDA_CUDNN_TF32

`torch.backends.cudnn.allow_tf32` 默认为 `False`。设置该环境变量时控制 `torch.backends.cudnn.allow_tf32` 默认为 `True`。

- 设置：`export PYTORCH_ALLOW_CUDA_CUDNN_TF32=1`
- 取消设置：`unset PYTORCH_ALLOW_CUDA_CUDNN_TF32`

5. 精度对比工具

PyTorch 精度对比工具支持在 PyTorch 脚本中插入接口，用来记录和保存 PyTorch 的 Python op API 的输入输出 tensor 信息，并提供接口比较不同 device 输出的 tensor 的精度异同，从而方便在复杂网络中定位到 MXMACA 算子相对 CUDA 算子哪个算子产生了较大的误差及误差是多少。

5.1 接口介绍

5.1.1 start_record

```
start_record(enabled: bool, output_dir: str = "./record_dir", record_level:
int = 2, record_stack = True, record_input: bool = True, record_output: bool
= True, op_range: List[str] = [], op_list: List[str] = [], skip = 0,
process_group = None, ranks = []) -> None
```

功能描述：初始化精度工具，从 start_record 开启之后的 Python 算子会被 record。

参数说明：

- `enabled`

  bool 类型，控制是否启用精度工具，初始化工具配置，默认值为 `True`。

- `output_dir`

  string 类型，record 数据存储路径，默认为存储在当前路径的 **record_dir** 目录。

- `record_level`

  int 类型，控制 dump 算子的输入和输出信息的详细程度，取值包括 `2`、`1` 和 `0`，默认为 `2`。

  - Level 2：表示 dump 该算子输入输出的所有信息，包含 Tensor 的完整信息。
  - Level 1：表示不会 dump Tensor 的完整信息，只包含 Tensor 的概述信息（我们定义的 TensorSummary 类型包含了 Tensor 的数据类型、形状、步长、元素最大值和最小值）。
  - Level 0：表示不会 dump 算子的输入输出信息。

- `record_stack`

  bool 类型，是否 record 算子的 Python 端调用栈信息，默认为 `True`。反向算子是 hook 函数的调用栈，随算子的输入或者输出信息一起保存到一个文件。

- `record_input`

  bool 类型，是否 record 算子的输入信息，默认为 `True`。

- `record_output`

  bool 类型，是否 record 算子的输出信息，默认为 `True`。

- op_range

  List[str]类型，控制 record 信息范围。只能传空列表、包含 Python 端 API 信息的字符串或者 None，长度为 2 的列表，例如：

  ["torch.nn.Conv1d.6.fwd", "torch.nn.AdaptiveAvgPool2d.256.bwd"] 、 [None, "torch.nn.AdaptiveAvgPool2d.256.bwd"]、["torch.nn.Conv1d.6.fwd", None]。

  None 表示不限制 record 的开始或者结束边界。

  字符串由 3 部分组成：算子的 API 官方名称、按照执行时间顺序对应的是第几个前向算子、前向（fwd）还是反向（bwd）。该控制范围前闭后开，默认为空列表，表示不限制 record 范围。

- p_list

  List[str]类型，控制 record 的具体前反向算子。字符串命名规则同 op_range，并且支持指定某一类 API，例如：api_list=["relu"] 会 record 包含 torch.relu、torch.relu_、torch.Tensor.relu、torch.nn.ReLU 和 torch.nn.functional.relu_ 等前反向算子。默认为空列表，表示不限制算子。

- skip

  int 类型，表示每间隔 skip 次 op 后 record 一次。默认为 0。

- process_group

  torch.distributed.ProcessGroup 类型，表示需要 record 的进程组。默认为 None，表示只会 record 默认进程组，如果不是分布式场景，不需要设置该参数。有关进程组的详细内容，参见 PyTorch 官方分布式模块相关文档。

- ranks

  List[int]类型，表示需要 record 进程组里面的哪些进程。默认为空列表，表示不限制。

5.1.2 record

```
record(enabled: bool, output_dir: str = "./record_dir", record_level: int =
2, record_stack = True, record_input: bool = True, record_output: bool =
True, op_range: List[str] = [], op_list: List[str] = [], skip = 0,
process_group = None, ranks = []) -> None
```

功能描述：上下文管理器，方便控制 record 范围，功能同 start_record。

参数说明：同 start_record。

5.1.3 record_switch

功能描述：开关 record 功能。

参数说明：

- enabled

  bool 类型，是否开启 record 数据，无默认值。

5.1.4 gen_report

```
gen_report(dir0: str, dir1: str, output_dir: str = "./") -> None
```

功能描述：输入 record 生成的算子 tensor 目录，生成逐 tensor 的误差、tensor 的概述信息和相关调用栈信息的 excel 表格报告。

参数说明：

- `dir0`

  string 类型，device0 的 record 数据路径，无默认值。

- `dir1`

  string 类型，device1 的 record 数据路径，无默认值。

- `output_dir`

  string 类型，比较结果的输出目录。默认为输出到当前目录。

5.2 生成文件介绍

PyTorch 精度工具，当开启 `start_record` 后会在指定的 output_dir 下生成 pt 文件，使用 `gen_report` 分析不同 device 的 output_dir 可以生成对比的 excel 表格。

- pt 文件：pt 文件有自己的命名方式，例如：
  - torch.nn.Conv1d.6.fwd 指的是前向第 6 个算子，这个算子是 torch.nn.Conv1d
  - torch.nn.AdaptiveAvgPool2d.256.bwd，指的是反向第 256 个算子，算子是 torch.nn.AdaptiveAvgPool2d
- excel 表格：如下图，生成的 excel 表格中包括 device0 和 device1 对应算子输入输出 tensor 的数据类型，shape，最大最小值以及 diff1（相对误差）diff2（标准差），和 op 的 Python 调用栈。

Dev0 Name	Dev1 Name	Dev0 Tensor Dtype	Dev1 Tensor Dtype	Dev0 Tensor	Dev1 Tensor S	Dev0 Tensor Max	Dev1 Tensor Max	Dev0 Tensor Min	Dev1 Tensor Min	Diff2 Error	Diff1 Error	Dev0 Stack	Dev1 Stack
torch.Tensor.uniform_.0.fwd.inpu	torch.Tensor.uniform_.0.fwd.inpu	torch.float16	torch.float16	(4, 1, 5)	(4, 1, 5)	nan	0	nan	0	nan	nan	['	File "test_conv1d.py", line
torch.Tensor.uniform_.0.fwd.out	torch.Tensor.uniform_.0.fwd.outp	torch.float16	torch.float16	(4, 1, 5)	(4, 1, 5)	0.407	0.373	-0.4456	-0.4202	1.3601	1.2141	['	File "test_conv1d.py", line
torch.Tensor.uniform_.1.fwd.inpu	torch.Tensor.uniform_.1.fwd.inpu	torch.float16	torch.float16	(4,)	(4,)	nan	nan	nan	nan	nan	nan	['	File "test_conv1d.py", line
torch.Tensor.uniform_.1.fwd.out	torch.Tensor.uniform_.1.fwd.outp	torch.float16	torch.float16	(4,)	(4,)	0.4294	0.4316	-0.1	-0.4009	1.1667	1.3663	['	File "test_conv1d.py", line
torch.Tensor.clone.2.fwd.input.0	torch.Tensor.clone.2.fwd.input.0	torch.float16	torch.float16	(4, 1, 5)	(4, 1, 5)	0.407	0.373	-0.4456	-0.4202	1.3601	1.2141	['	File "test_conv1d.py", line
torch.Tensor.clone.2.fwd.output	torch.Tensor.clone.2.fwd.output.(	torch.float16	torch.float16	(4, 1, 5)	(4, 1, 5)	0.407	0.373	-0.4456	-0.4202	1.3601	1.2141	['	File "test_conv1d.py", line
torch.Tensor.clone.3.fwd.input.0	torch.Tensor.clone.3.fwd.input.0	torch.float16	torch.float16	(4,)	(4,)	0.4294	0.4316	-0.1	-0.4009	1.1667	1.3663	['	File "test_conv1d.py", line
torch.Tensor.clone.3.fwd.output	torch.Tensor.clone.3.fwd.output.(	torch.float16	torch.float16	(4,)	(4,)	0.4294	0.4316	-0.1	-0.4009	1.1667	1.3663	['	File "test_conv1d.py", line
torch.Tensor.to.4.fwd.input.0	torch.Tensor.to.4.fwd.input.0	torch.float16	torch.float16	(4, 1, 5)	(4, 1, 5)	0.407	0.373	-0.4456	-0.4202	1.3601	1.2141	['	File "test_conv1d.py", line
torch.Tensor.to.4.fwd.output.0	torch.Tensor.to.4.fwd.output.0	torch.float16	torch.float16	(4, 1, 5)	(4, 1, 5)	0.407	0.373	-0.4456	-0.4202	1.3601	1.2141	['	File "test_conv1d.py", line
torch.Tensor.to.5.fwd.input.0	torch.Tensor.to.5.fwd.input.0	torch.float16	torch.float16	(4,)	(4,)	0.4294	0.4316	-0.1	-0.4009	1.1667	1.3663	['	File "test_conv1d.py", line
torch.Tensor.to.5.fwd.output.0	torch.Tensor.to.5.fwd.output.0	torch.float16	torch.float16	(4,)	(4,)	0.4294	0.4316	-0.1	-0.4009	1.1667	1.3663	['	File "test_conv1d.py", line
torch.nn.Conv1d.6.fwd.input.0	torch.nn.Conv1d.6.fwd.input.0	torch.float16	torch.float16	(4, 1, 5)	(4, 1, 5)	0.407	0.373	-0.4456	-0.4202	1.3601	1.2141	['	File "test_conv1d.py", line
torch.nn.Conv1d.6.fwd.input.1	torch.nn.Conv1d.6.fwd.input.1	torch.float16	torch.float16	(4,)	(4,)	0.4294	0.4316	-0.1	-0.4009	1.1667	1.3663	['	File "test_conv1d.py", line
torch.nn.Conv1d.6.fwd.input.2	torch.nn.Conv1d.6.fwd.input.2	torch.float16	torch.float16	(256, 1, 360(	(256, 1, 3600)	3.9043	3.9043	-3.9043	-3.9043	1.4133	1.4132	['	File "test_conv1d.py", line
torch.nn.Conv1d.6.fwd.output.0	torch.nn.Conv1d.6.fwd.output.0	torch.float16	torch.float16	(256, 4, 360(	(256, 4, 3600)	3.9336	3.4746	-3.4629	-3.3262	1.2995	1.2967	['	File "test_conv1d.py", line

图 5-1 生成文件

5.3 使用流程

1. 分别在 device0（如 CUDA-GPU 或 CPU）和 device1（如 MACA-GPU）上运行精度比较工具，保存网络训练中间数据。

2. 再次运行精度比较工具比对 device0 和 device1 保存下来的数据，计算误差并输出比对结果。

3. `gen_report` 分析比对结果，分析的过程类似二分法，设置一个比较大的 skip 数（加快速度），找出导致误差大的算子。如果没有找到误差大的算子，则需要重新调整 record 范围减小 skip 数，再次 record。如果找到误差大的算子，说明这个 op_range 存在误差大的算子，但由于 skip 存在这个误差大的算子只是 skip 的区间末尾，所以应该将 skip 设置为 0 找到具体某个 op 导致的精度误差。

下面是一个简单的测试用例，`start_record` 和 `record_switch(False)` 之间的 op 将被 record：

```
import torch
import torch.nn as nn
import copy
from maca_tools import accuracy

dtype = torch.float16
shape, out_c, k, s, p, d, g = (256, 1, 3600), 4, 5, 1, 2, 1, 1
input = torch.randn(shape, dtype = dtype, device="cpu")
input_d = input.clone().cuda()
m = nn.Conv1d(shape[1], out_c, k, stride = s, padding = p, dilation = d,
groups = g, dtype = dtype)
m_d = copy.deepcopy(m)
m_d = m_d.to("cuda")
accuracy.start_record(record_level=2)
output = m_d(input_d)
accuracy.record_switch(False)
```

如果执行 kernel 过多，则需要重新调整 record 范围减小 skip 数再次 record。

下述案例为 skip=10，表示每间隔 10 次 op 后 record 一次，统计到 torch.nn.Conv1d.6.fwd 结束。

```
import torch
import torch.nn as nn
import copy
from maca_tools import accuracy

dtype = torch.float16
shape, out_c, k, s, p, d, g = (256, 1, 3600), 4, 5, 1, 2, 1, 1
input = torch.randn(shape, dtype = dtype, device="cpu")
input_d = input.clone().cuda()
m = nn.Conv1d(shape[1], out_c, k, stride = s, padding = p, dilation = d,
groups = g, dtype = dtype)
m_d = copy.deepcopy(m)
m_d = m_d.to("cuda")
accuracy.start_record(record_level=2, skip=10, op_range=[None,
"torch.nn.Conv1d.6.fwd"])
output = m_d(input_d)
accuracy.record_switch(False)
```

在 CUDA 运行时，安装 maca_tools 的方法如下：

1. 通过以下命令解压 MXMACA torch 的 wheel 包，在当前目录下会得到 **maca_tools** 的文件夹。

```
unzip torch-2.0.0*.whl
```

2. 在 CUDA 环境下，将上述解压获得的 **maca_tools** 拷贝到环境中 wheel 包存放的目录。

```
cp -r maca_tools/ $(pip show torch | grep Location | awk '{print $2}')
```

3. 其余部分 CUDA 运行方式与 MXMACA 相同，同上所述。

5.4 注意事项

- 保证两次 record 过程调用的算子数量和算子顺序一致。
- 为使比较结果更准确，需提前消除网络计算中的随机性和因算子算法差异导致的比对误差等干扰因素。
- 如果存在有算子 API 在工具初始化之前被导入，可能会导致错过 Python 接口劫持时机，进而导致这些 Python 接口不会被 record。
- 算子可能 record 不完整，具体表现在：该工具只能 record Python 侧 op，如果这个 Python op 对应的是多个 C++拼接实现的算子，只是 record 最后那个 C++算子的输出。
- 目前不能单独 record 反向算子的相关信息，必须带上相应的前向算子。
- 开启 record 功能的网络性能会有一定程度下降。保存信息庞大，用户需预留足够空间存放。
- 本工具跟 PyTorch 版本绑定，不保证其他版本的 PyTorch 也能使用。
- 分布式多卡训练时，start_record 需要在 init_process_group 初始化之后执行。
- 网络脚本中如果直接调用 torch._C._VariableFunctions.xxx 或者 torch._C._TensorBase.xxx 这两种 PyTorch 内部使用的接口，目前无法 record。
- 暂不支持 record 少数输出不在 API 返回值中的原位算子的输出，如： torch.Tensor.__setitem__、torch._amp_foreach_non_finite_check_and_unscale_。
- 暂不支持 record 少数不涉及计算的 tensor 构造类 API，包括 torch.tensor、torch.empty、torch.empty_like、torch.empty_strided、torch.empty_quantized。
- 暂不支持 record torch.Tensor.__getitem__。
- 不支持 record 工具初始化所在进程的子进程和子线程的算子的输入输出信息。
- 不支持导出 torch.Generator 类型的算子参数的完整信息。
- 不支持 record 稀疏 Tensor、量化 Tensor、NestedTensor 及相关算子的输入输出信息。
- 不支持 record 通信原语。
- 不支持 record torch.jit.script 后的 ScriptModule 和 ScriptFunction 的输入和输出信息。
- 由于 torch.Tensor.uniform_.fwd 算子的 input 为随机值，因此可以忽略。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Dev0 Name	Dev1 Nan	Dev0 Tens	Dev1 Tens	Dev0 Tens	Dev1 Tens	Dev0 Tens	Dev1 Tens	Dev0 Tens	Dev1 Tens	Diff2 Error	Diff1 Error	Dev0 Stac	Dev1 Stack	
784	torch.Tensor.reshape.380.fwd.input.0	torch.Tens	torch.float	torch.float	(32, 1)	(32, 1)	0	0	0	0	0	0	[' File "de	[' File "detect.py",	
785	torch.Tensor.reshape.380.fwd.output.0	torch.Tens	torch.float	torch.float	(32,)	(32,)	0	0	0	0	0	0	[' File "de	[' File "detect.py",	
786	torch.Tensor.__add__.381.fwd.input.0	torch.Tens	torch.float	torch.float	(32,)	(32,)	0	0	0	0	0	0	[' File "de	[' File "detect.py",	
787	torch.Tensor.__add__.381.fwd.input.1	torch.Tens	torch.float	torch.float	(32,)	(32,)	2.8777	2.8777	-0.9988	-0.9988	0	0	[' File "de	[' File "detect.py",	
788	torch.Tensor.__add__.381.fwd.output.0	torch.Tens	torch.float	torch.float	(32,)	(32,)	2.8777	2.8777	-0.9988	-0.9988	0	0	[' File "de	[' File "detect.py",	
789	torch.Tensor.copy_.382.fwd.input.0	torch.Tens	torch.float	torch.float	(32,)	(32,)	0.1187	0.1187	-0.1176	-0.1176	0	0	[' File "de	[' File "detect.py",	
790	torch.Tensor.copy_.382.fwd.input.1	torch.Tens	torch.float	torch.float	(32,)	(32,)	2.8777	2.8777	-0.9988	-0.9988	0	0	[' File "de	[' File "detect.py",	
791	torch.Tensor.copy_.382.fwd.output.0	torch.Tens	torch.float	torch.float	(32,)	(32,)	2.8777	2.8777	-0.9988	-0.9988	0	0	[' File "de	[' File "detect.py",	
792	torch.Tensor.uniform_.383.fwd.input.0	torch.Tens	torch.float	torch.float	(64, 64, 1,	(64, 64, 1,	2.34E+37	nan	######	nan	nan	nan	[' File "de	[' File "detect.py",	
793	torch.Tensor.uniform_.383.fwd.output.0	torch.Tens	torch.float	torch.float	(64, 64, 1,	(64, 64, 1,	0.1248	0.1248	-0.125	-0.125	0	0	[' File "de	[' File "detect.py",	
794	torch.Tensor.uniform_.384.fwd.input.0	torch.Tens	torch.float	torch.float	(64,)	(64,)	1.53E+16	2.90E+26	######	0	nan	1.0018	[' File "de	[' File "detect.py",	
795	torch.Tensor.uniform_.384.fwd.output.0	torch.Tens	torch.float	torch.float	(64,)	(64,)	0.1161	0.1161	-0.1219	-0.1219	0	0	[' File "de	[' File "detect.py",	
796	torch.Tensor.to.385.fwd.input.0	torch.Tens	torch.float	torch.float	(64, 64, 1,	(64, 64, 1,	0.1248	0.1248	-0.125	-0.125	0	0	[' File "de	[' File "detect.py",	
797	torch.Tensor.to.385.fwd.output.0	torch.Tens	torch.float	torch.float	(64, 64, 1,	(64, 64, 1,	0.1248	0.1248	-0.125	-0.125	0	0	[' File "de	[' File "detect.py",	
798	torch.Tensor.to.386.fwd.input.0	torch.Tens	torch.float	torch.float	(64,)	(64,)	0.1161	0.1161	-0.1219	-0.1219	0	0	[' File "de	[' File "detect.py",	
799	torch.Tensor.to.386.fwd.output.0	torch.Tens	torch.float	torch.float	(64,)	(64,)	0.1161	0.1161	-0.1219	-0.1219	0	0	[' File "de	[' File "detect.py",	

图 5-2 input 示例

6. Resnet50 运行示例

mcPyTorch 安装包中提供了一份可供直接运行的 Resnet50 简单示例，用户可参考以下命令运行：

```
sample_dir="$(pip show torch | grep Location | awk '{print $2}')/torch/share/sample/resnet50"
cd ${sample_dir}
bash run_resnet50.sh
```

运行结束后，会打印 `Resnet50 test checkout finish!`。

声明