摩尔线程
MOORE THREADS

摩尔线程 Torch_MUSA 开发者手册

版本 ***0.1.0***

2023 年 07 月 25 日

摩尔线程
MOORE THREADS

目录

1 版权声明

版权章节待完善。

- 版权声明
- © 2023

2 前言

2.1 版本记录

表 2.1: 版本记录

文档名称	摩尔线程 Torch_MUSA 开发者手册
版本号	V 0.1.0
作者	Moore Threads
修改日期	2023 年 07 月 25 日

2.2 更新历史

- **V0.1.0**

  更新时间: 2023.05.29

  更新内容:

  – 完成初版 Torch_MUSA 开发者手册。

摩尔线程
MOORE THREADS

3 简介

3.1 MUSA 概述

MUSA (Metaverse Unified System Architecture) 是摩尔线程公司为摩尔线程 GPU 推出的一种通用并行计算平台和编程模型。它提供了 GPU 编程的简易接口，用 MUSA 编程可以构建基于 GPU 计算的应用程序，利用 GPUs 的并行计算引擎来更加高效地解决比较复杂的计算难题。同时摩尔线程还推出了 MUSA 工具箱（MUSAToolkits），工具箱中包括 GPU 加速库，运行时库，编译器，调试和优化工具等。MUSAToolkits 为开发人员在摩尔线程 GPU 上开发和部署高性能异构计算程序提供软件环境。

关于 MUSA 软件栈的更多内容，请参见 MUSA 官方文档。

3.2 PyTorch 概述

PyTorch 是一款开源的深度学习编程框架，可以用于计算机视觉，自然语言处理，语音处理等领域。PyTorch 使用动态计算，这在构建复杂架构时提供了更大的灵活性。PyTorch 使用核心 Python 概念，如类、结构和条件循环，因此理解起来更直观，编程更容易。此外，PyTorch 还具有可以轻松扩展、快速实现、生产部署稳定性强等优点。

关于 PyTorch 的更多内容，请参见 PyTorch 官方文档。

3.3 torch_musa 概述

为了摩尔线程 GPU 能支持开源框架 PyTorch，摩尔线程公司开发了 torch_musa。在 PyTorch v2.0.0 基础上，torch_musa 以插件的形式来支持摩尔线程 GPU，最大程度与 PyTorch 代码解耦，便于代码维护与升级。torch_musa 利用 PyTorch 提供的第三方后端扩展接口，将摩尔线程高性能计算库动态注册到 PyTorch 上，从而使得 PyTorch 框架能够利用摩尔线程显卡的高性能计算单元。利用摩尔线程显卡 CUDA 兼容的特性，torch_musa 内部引入了 cuda 兼容模块，使得 PyTorch 社区的 CUDA kernels 经过 porting 后就可以运行在摩尔线程显卡上，而且 CUDA Porting 的工作是在编译 torch_musa 的过程中自动进行，这大幅降低了 torch_musa 算子适配的成本，提高模型开发效率。同时，torch_musa 在 Python 前端接口与 PyTorch 社区 CUDA 接口形式上基本保持一致，这极大地降低了用户的学习成本和模型的迁移成本。

本手册主要介绍了基于 MUSA 软件栈的 torch_musa 开发指南。

3.4 torch_musa 核心代码目录概述

- torch_musa/tests 测试文件
- torch_musa/core 主要包含 Python module，提供 Python 前端的接口。
- torch_musa/csrc c++ 侧实现代码
  - csrc/aten 提供 C++ Tensor 库，包括 MUDNN 算子适配，CUDA-Porting 算子适配等等
  - csrc/core 提供核心功能库，包括设备管理，内存分配管理，Stream 管理，Events 管理等

摩尔线程
MOORE THREADS

4 编译安装

注意：

编译安装前，需要安装 MUSAToolkits 软件包，MUDNN 库，muThrust 库，muAlg 库，muRAND 库，muSPARSE 库。具体安装步骤，请参见相应组件的安装手册。

4.1 依赖环境

- Python == 3.8 或者 Python == 3.9。
- 摩尔线程 MUSA 软件包，推荐版本如下：
  - MUSA 驱动 rc1.4.1
  - MUSAToolkits Release1.4.1
  - MUDNN rc1.4.1
  - muAlg_dev-0.1.1-Linux.deb
  - muRAND_dev1.0.0.tar.gz
  - muSPARSE_dev0.1.0.tar.gz
  - muThrust_dev-0.1.1-Linux.deb
  - Docker Container Toolkits(https://mcconline.mthreads.com/software)

4.2 编译流程

1. 向 PyTorch 源码打 patch
2. 编译 PyTorch
3. 编译 torch_musa

torch_musa 是在 PyTorch v2.0.0 基础上以插件的方式来支持摩尔线程显卡。开发时涉及到对 PyTorch 源码的修改，目前是以打 patch 的方式实现的。PyTorch 社区正在积极支持第三方后端接入，https://github.com/pytorch/pytorch/issues/98406 这个 issue 下有相关 PR。我们也在积极向 PyTorch 社区提交 PR，避免在编译过程中向 PyTorch 打 patch。

4.3 开发 Docker

为了方便开发者开发 torch_musa，我们提供了开发用的 docker image，参考命令：

```
docker run -it --name=torch_musa_dev --env MTHREADS_VISIBLE_DEVICES=all --shm-size=80g sh-
↪harbor.mthreads.com/mt-ai/musa-pytorch-dev:latest /bin/bash
```

注意：

使用 docker 时，请务必提前安装 mt-container-toolkit(https://mcconline.mthreads.com/software/1?id=1)，并且在启动 docker container 时添加选项“--env MTHREADS_VISIBLE_DEVICES=all”，否则在 docker container 内部无法使用 torch_musa。

4.4 编译步骤

4.4.1 设置环境变量

```
export MUSA_HOME=path/to/musa_libraries(including musa_toolkits, mudnn and so on) # defalut␣
↪value is /usr/local/musa/
export LD_LIBRARY_PATH=$MUSA_HOME/lib:$LD_LIBRARY_PATH
export PYTORCH_REPO_PATH=path/to/PyTorch source code
# if PYTORCH_REPO_PATH is not set, PyTorch-v2.0.0 will be downloaded outside this directory␣
↪automatically when building with build.sh
```

4.4.2 使用脚本一键编译（推荐）

```
cd torch_musa
bash scripts/update_daily_mudnn.sh # update daily mudnn lib if needed
bash build.sh   # build original PyTorch and torch_musa from scratch

# Some important parameters are as follows:
bash build.sh --torch  # build original PyTorch only
bash build.sh --musa   # build torch_musa only
bash build.sh --fp64   # compile fp64 in kernels using mcc in torch_musa
bash build.sh --debug  # build in debug mode
bash build.sh --asan   # build in asan mode
bash build.sh --clean  # clean everything built
```

在初次编译时，需要执行 `bash build.sh`（先编译 PyTorch，再编译 torch_musa）。在后续开发过程中，如果不涉及对 PyTorch 源码的修改，那么执行 `bash build.sh -m`（仅编译 torch_musa）即可。

4.4.3 分步骤编译

如果不想使用脚本编译，那么可以按照如下步骤逐步编译。

1. 在 PyTorch 打 patch

```
# 请保证 PyTorch 源码和 torch_musa 源码在同级目录或者 export PYTORCH_REPO_PATH=path/to/PyTorch 指向 PyTorch 源码
bash build.sh --only-patch
```

2. 编译 PyTorch

```
cd pytorch
pip install -r requirements.txt
python setup.py install
# debug mode: DEBUG=1 python setup.py install
# asan mode:  USE_ASAN=1 python setup.py install
```

3. 编译 torch_musa

```
cd torch_musa
pip install -r requirements.txt
python setup.py install
# debug mode: DEBUG=1 python setup.py install
# asan mode:  USE_ASAN=1 python setup.py install
```


摩尔线程
MOORE THREADS

5 快速入门

注解：

使用 torch_musa 时，需要先导入 torch 包（import torch）和 torch_musa 包 (import torch_musa)。

5.1 常用环境变量

开发 torch_musa 过程中常用环境变量如下表所示：

表 5.1: 常用环境变量

环境变量示例	所属组件	功能说明
export MUDNN_LOG_LEVEL=INFO	MUDNN	使能 MUDNN 算子库调用的 log
export MUSA_VISIBLE_DEVICES=0,1,2,3	Driver	控制当前可见的显卡序号
export MUSA_LAUNCH_BLOCKING=1	Driver	驱动以同步模式下发 kernel，即当前 kernel 执行结束后再下发下一个 kernel

5.2 常用 api 示例代码

```
import torch
import torch_musa

torch.musa.is_available()
torch.musa.device_count()

a = torch.tensor([1.2, 2.3], dtype=torch.float32, device='musa')
b = torch.tensor([1.8, 1.2], dtype=torch.float32, device='musa')
c = a + b

torch.musa.synchronize()
```

```
with torch.musa.device(0):
    assert torch.musa.current_device() == 0

if torch.musa.device_count() > 1:
    torch.musa.set_device(1)
    assert torch.musa.current_device() == 1
    torch.musa.synchronize("musa:1")
```

torch_musa 中 python api 基本与 PyTorch 原生 api 接口保持一致，极大降低了新用户的学习成本。

5.3 推理示例代码

```
import torch
import torch_musa
import torchvision.models as models

model = models.resnet50().eval()
x = torch.rand((1, 3, 224, 224), device="musa")
model = model.to("musa")
# Perform the inference
y = model(x)
```

5.4 训练示例代码

```
import torch
import torch_musa
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

## 1. prepare dataset
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 4
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
```



```
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)
classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
device = torch.device("musa")

## 2. build network
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net().to(device)

## 3. define loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

## 4. train
for epoch in range(2):  # loop over the dataset multiple times
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
```



```
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs.to(device))
        loss = criterion(outputs, labels.to(device))
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
            running_loss = 0.0

print('Finished Training')

PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)

net.load_state_dict(torch.load(PATH))

## 5. test
correct = 0
total = 0
# since we're not training, we don't need to calculate the gradients for our outputs
with torch.no_grad():
    for data in testloader:
        images, labels = data
        # calculate outputs by running images through the network
        outputs = net(images.to(device))
        # the class with the highest energy is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels.to(device)).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 * correct // total} %')
```


5.5 C++ 部署示例代码

```cpp
#include <torch/script.h>
#include <torch_musa/csrc/core/Device.h>
#include <iostream>
#include <memory>

int main(int argc, const char* argv[]) {
  // Register 'musa' for PrivateUse1 as we save model with 'musa'.
  c10::register_privateuse1_backend("musa");

  torch::jit::script::Module module;
  // Load model which saved with torch jit.trace or jit.script.
  module = torch::jit::load(argv[1]);

  std::vector<torch::jit::IValue> inputs;
  // Ready for input data.
  torch::Tensor input = torch::rand({1, 3, 224, 224}).to("musa");
  inputs.push_back(input);

  // Model execute.
  at::Tensor output = module.forward(inputs).toTensor();

  return 0;
}
```

详细用法请参考 examples/cpp[1] 下内容。

[1] https://github.mthreads.com/mthreads/torch_musa/tree/main/examples/cpp

6 算子支持

本节主要介绍如何在 torch_musa 中适配一个新算子，算子实现后端包括 MUDNN 算子库和 CUDA-Porting kernels。

6.1 何时需要适配新算子?

以“tril”算子为例，当我们的测试代码有如下报错 log，则说明 torch_musa 中没有适配“tril”算子。

```
import torch
import torch_musa
input_data = torch.randn(3, 3, device="musa")
result = torch.tril(input_data)
```

```
Traceback (most recent call last):
  File "test.py", line 5, in <module>
    result = torch.tril(input_data)
NotImplementedError: Could not run 'aten::tril.out' with arguments from the 'musa' backend. This could be because the operator doesn't exist for this backend, or was omitted during the selective/custom build process (if using custom build). If you are a Facebook employee using PyTorch on mobile, please visit https://fburl.com/ptmfixes for possible resolutions. 'aten::tril.out' is only available for these backends: [CPU, Meta, BackendSelect, Python, FuncTorchDynamicLayerBackMode, Functionalize, Named, Conjugate, Negative, ZeroTensor, ADInplaceOrView, AutogradOther, AutogradCPU, AutogradCUDA, AutogradHIP, AutogradXLA, AutogradMPS, AutogradIPU, AutogradXPU, AutogradHPU, AutogradVE, AutogradLazy, AutogradMeta, AutogradMTIA, AutogradPrivateUse1, AutogradPrivateUse2, AutogradPrivateUse3, AutogradNestedTensor, Tracer, AutocastCPU, AutocastCUDA, FuncTorchBatched, FuncTorchVmapMode, Batched, VmapMode, FuncTorchGradWrapper, PythonTLSSnapshot, FuncTorchDynamicLayerFrontMode, PythonDispatcher].

CPU: registered at /home/pytorch/build/aten/src/ATen/RegisterCPU.cpp:31034 [kernel]
Meta: registered at /dev/null:219 [kernel]
BackendSelect: fallthrough registered at /home/pytorch/aten/src/ATen/core/BackendSelectFallbackKernel.cpp:3 [backend fallback]
Python: registered at /home/pytorch/aten/src/ATen/core/PythonFallbackKernel.cpp:144 [backend fallback]
FuncTorchDynamicLayerBackMode: registered at /home/pytorch/aten/src/ATen/functorch/DynamicLayer.cpp:491 [backend fallback]
Functionalize: registered at /home/pytorch/build/aten/src/ATen/RegisterFunctionalization_1.cpp:23013 [kernel]
Named: registered at /home/pytorch/aten/src/ATen/core/NamedRegistrations.cpp:7 [backend fallback]
Conjugate: registered at /home/pytorch/aten/src/ATen/ConjugateFallback.cpp:17 [backend fallback]
```

6.2 如何适配新算子

6.2.1 先注册新算子

算子实现的注册可以参考 PyTorch 官方文档 https://pytorch.org/tutorials/advanced/dispatcher.html，也可以参考 PyTorch 框架中 CUDA 后端的注册代码。在编译完 PyTorch 代码后会生成下图中的文件，PyTorch 在该文件中实现了 CUDA 后端实现的注册。

```
m.impl("bitwise_left_shift_.Tensor", TORCH_FN(wrapper_CUDA_bitwise_left_shift__Tensor))
m.impl("__rshift__.Scalar",
TORCH_FN(wrapper_CUDA_Scalar___rshift__));
m.impl("__irshift__.Scalar",
TORCH_FN(wrapper_CUDA_Scalar___irshift__));
m.impl("__rshift__.Tensor",
TORCH_FN(wrapper_CUDA_Tensor___rshift__));
m.impl("__irshift__.Tensor",
TORCH_FN(wrapper_CUDA_Tensor___irshift__));
m.impl("bitwise_right_shift.Tensor", TORCH_FN(wrapper_CUDA_bitwise_right_shift_Tensor))
m.impl("bitwise_right_shift.Tensor_out", TORCH_FN(wrapper_CUDA_bitwise_right_shift_out_
m.impl("bitwise_right_shift_.Tensor", TORCH_FN(wrapper_CUDA_bitwise_right_shift__Tensor
m.impl("tril", TORCH_FN(wrapper_CUDA_tril));
m.impl("tril.out", TORCH_FN(wrapper_CUDA_tril_out_out));
m.impl("tril_", TORCH_FN(wrapper_CUDA_tril_));
m.impl("triu", TORCH_FN(wrapper_CUDA_triu));
m.impl("triu.out", TORCH_FN(wrapper_CUDA_triu_out_out));
m.impl("triu_", TORCH_FN(wrapper_CUDA_triu_));
m.impl("digamma", TORCH_FN(wrapper_CUDA_digamma));
m.impl("digamma.out", TORCH_FN(wrapper_CUDA_digamma_out_out));
m.impl("digamma_", TORCH_FN(wrapper_CUDA_digamma_));
m.impl("lerp.Scalar", TORCH_FN(wrapper_CUDA_lerp_Scalar));
m.impl("lerp.Scalar_out", TORCH_FN(wrapper_CUDA_lerp_out_Scalar_out));
m.impl("lerp_.Scalar", TORCH_FN(wrapper_CUDA_lerp__Scalar));
m.impl("lerp.Tensor", TORCH_FN(wrapper_CUDA_lerp_Tensor));
m.impl("lerp.Tensor_out", TORCH_FN(wrapper_CUDA_lerp_out_Tensor_out));
m.impl("lerp_.Tensor", TORCH_FN(wrapper_CUDA_lerp__Tensor));
m.impl("addbmm",
TORCH_FN(wrapper_CUDA__addbmm));
m.impl("addbmm.out",
TORCH_FN(wrapper_CUDA_out_addbmm_out));
m.impl("addbmm_",
TORCH_FN(wrapper_CUDA__addbmm_));
m.impl("random_.from",
TORCH_FN(wrapper_CUDA_from_random_));
m.impl("random_.to",
TORCH_FN(wrapper_CUDA_to_random_));
m.impl("random_",
"pytorch/build/aten/src/ATen/RegisterCUDA.cpp" 50827L, 2715252B
```

同理，我们也需要给“tril”算子为 MUSA 后端实现注册。部分代码如下所示：

```cpp
#include <torch/library.h>

namespace at {
namespace musa {

TORCH_LIBRARY_IMPL(aten, PrivateUse1, m) {
  m.impl("tril", Tril);  // 'Tril' is a function implemented somewhere
}

} // namespace musa
} // namespace at
```


PyTorch 社区推荐使用 PrivateUse1 作为第三方扩展后端的 key，所以我们这里复用了 PrivateUse1。

6.2.2 利用 MUDNN 实现算子

如果 MUDNN 算子库支持了该算子，那么需要以 MUDNN 算子库作为后端来适配该新算子。使用 MUDNN 适配新算子的主要步骤如下：

1. tensor 的数据类型和 device 类型检查;
2. 添加 DeviceGuard;
3. 目前大部分 MUDNN 算子只支持连续的 tensor，因此在 `createMUTensor` 前需要将其转换为连续 tensor（如果该 tensor 不连续，此操作将会产生 copy 耗时）;
4. 创建 muTensor;
5. 调用 MUDNN 的 op 实现接口;

以 `addcdiv.out` 算子为例，部分代码如下：

```
#include <mudnn.h>

Tensor& AddcDivOut(const Tensor& base, const Tensor& tensor1,
                   const Tensor& tensor2, const Scalar& value, Tensor& out) {
  //1). check Dtype & device
  TORCH_CHECK(self.device().type() == kMUSA,
              "Device of input tensor of addcdiv must be MUSA, but now it is ",
              self.device());
  TORCH_CHECK(
      self.scalar_type() == at::ScalarType::Float,
      "Dtype of input tensor of addcdiv only support Float32, but now it is ",
      self.scalar_type());
     ....
  // 2).convert it to contiguous tensor
  Tensor tensor_cong = Contiguous(tensor1);
     ...
  // 3). create muTensor, which binds the two variables by address.
  muTensor musa_tensor1 = CreateMUTensor(tensor_cong);
  muTensor mu_out = CreateMUTensor(tensor_cong);     ....
  // 4). call musa op to implement the calculation.
  ::musa::dnn::Handle h;
  ::musa::dnn::Ternary mop;

  if (!alpha_scalar.equal(1)) {
    if (self.is_floating_point()) {
      CHECK_MUDNN_STATUS(mop.SetAlpha(alpha_scalar.toDouble()), "SetAlpha");
    } else {
```



```
      CHECK_MUDNN_STATUS(mop.SetAlpha(alpha_scalar.toLong()), "SetAlpha");
    }
  }
  CHECK_MUDNN_STATUS(mop.SetMode(TERNARY_MODE::ADDCDIV_ALPHA), "SetMode");
  CHECK_MUDNN_STATUS(mop.Run(h, om_mt, musa_base, musa_tensor1, musa_tensor2), "Run");

}

TORCH_LIBRARY_IMPL(aten, PrivateUse1, m){
   ...
   m.impl("addcdiv.out", &AddcDivOut);
}
```

通过 mudnn*.h 头文件可以查看到 MUDNN 算子库函数接口。默认 MUDNN 算子库的头文件会在 /usr/local/musa/include 目录下。

6.2.3 利用 CUDA-Porting 实现算子

如果该算子 MUDNN 算子库不支持，那么我们需要通过 CUDA-Porting kernels 作为后端来适配新算子。

首先介绍一下 CUDA-Porting 的流程：

1. 在 torch_musa/build 下新建目录（默认目录名是 torch_musa/build/generated_cuda_compatible）用来保存 CUDA-Porting 过程需要用到的文件。
2. 从 PyTorch 仓库中将 kernels 相关的 cu/cuh 文件以及 include 头文件复制到上一步新建目录中去。这些文件需要经过 CUDA-Porting 脚本的处理（torch_musa/torch_musa/tools/cuda_porting/cuda_porting.py）。
3. 运行 porting 工具。主要是一些字符串替换处理，如将 cudaMalloc 替换成 musaMalloc，cuda_fp16.h 替换成 musa_fp16.h 等。
4. 经过上述操作后，build/generated_cuda_compatible/aten/src/ATen/native/musa/会有很多****.mu 文件，这些 mu 文件就是我们适配时会用到的 kernels 文件。
5. 适配 CUDA-Porting 工具处理过的 kernels。

上述步骤 1，2，3，4 会在编译过程中自动完成，适配新算子关心的步骤 5 即可。有一点需要注意的是，在开发过程中引用的 PyTorch 头文件来自于 torch_musa/build/generated_cuda_compatible/include 目录，而不是系统下 PyTorch 安装目录下的头文件。

下面以两种典型算子为例，介绍如何利用 CUDA-Porting kernels 适配新算子。在开始适配之前，可以在 pytorch/build/aten/src/ATen/RegisterCUDA.cpp 文件中查看该算子在 CUDA 中的实现方式。

6.2.3.1 以 abs 算子为例

CUDA 中 abs 算子的部分适配代码如下：

```
at::Tensor & wrapper_CUDA_out_abs_out(const at::Tensor & self, at::Tensor & out) {
  // No device check
const OptionalDeviceGuard device_guard(device_of(self));
return at::native::abs_out(self, out);
}

******
m.impl("abs.out", TORCH_FN(wrapper_CUDA_out_abs_out));
```

如果该算子直接调用了 at::native 下面的函数接口，那么我们也这么做就可以了：

```
#include "torch_musa/csrc/core/MUSAGuard.h"
at::Tensor& MusaAbsout(const at::Tensor& self, at::Tensor& out) {
c10::musa::MUSAGuard device_gaurd(self.device());
return at::native::abs_out(self, out);
}

TORCH_LIBRARY_IMPL(aten, PrivateUse1, m) {
  m.impl("abs.out", &MusaAbsout);
}
```

这里的关键是 PyTorch 仓库提供了 DispatchStub 机制。我们在 CUDA-Porting 时，将 `REGISTER_CUDA_DISPATCH` 替换成 `REGISTER_MUSA_DISPATCH`，从而能实现根据 device 类型调用到 porting 后的 kernels。对这背后机制感兴趣的话，可以查看一下如下几个文件：

- abs_out 函数实现：https://github.com/pytorch/pytorch/blob/v2.0.0/aten/src/ATen/native/UnaryOps.cpp#L546
- abs_stub 注册：https://github.com/pytorch/pytorch/blob/v2.0.0/aten/src/ATen/native/cuda/AbsKernel.cu#L49
- DispatchStub 定义：https://github.com/pytorch/pytorch/blob/v2.0.0/aten/src/ATen/native/DispatchStub.h

6.2.3.2 以 tril 算子为例

CUDA 中 tril 算子的部分适配代码如下：

```
struct structured_tril_cuda_functional final : public at::native::structured_tril_cuda {
    void set_output_strided(
        int64_t output_idx, IntArrayRef sizes, IntArrayRef strides,
        TensorOptions options, DimnameList names
    ) override {
        auto current_device = guard_.current_device();
        if (C10_UNLIKELY(current_device.has_value())) {
          TORCH_INTERNAL_ASSERT(*current_device == options.device(),
            "structured kernels don't support multi-device outputs");
        } else {
          guard_.reset_device(options.device());
        }
        outputs_[output_idx] = create_out(sizes, strides, options);
        if (!names.empty()) {
          namedinference::propagate_names(*outputs_[output_idx], names);
        }
        // super must happen after, so that downstream can use maybe_get_output
        // to retrieve the output
    }
    void set_output_raw_strided(
        int64_t output_idx, IntArrayRef sizes, IntArrayRef strides,
        TensorOptions options, DimnameList names
    ) override {
        auto current_device = guard_.current_device();
        if (C10_UNLIKELY(current_device.has_value())) {
          TORCH_INTERNAL_ASSERT(*current_device == options.device(),
            "structured kernels don't support multi-device outputs");
        } else {
          guard_.reset_device(options.device());
        }
        outputs_[output_idx] = create_out(sizes, strides, options);
        if (!names.empty()) {
          namedinference::propagate_names(*outputs_[output_idx], names);
        }
        // super must happen after, so that downstream can use maybe_get_output
        // to retrieve the output
    }
    const Tensor& maybe_get_output(int64_t output_idx) override {
```

```
      return *outputs_[output_idx];
    }
    std::array<c10::ExclusivelyOwned<Tensor>, 1> outputs_;
    c10::cuda::OptionalCUDAGuard guard_;
};
at::Tensor wrapper_CUDA_tril(const at::Tensor & self, int64_t diagonal) {
c10::optional<Device> common_device = nullopt;
(void)common_device; // Suppress unused variable warning
  c10::impl::check_and_update_common_device(common_device, self, "wrapper_CUDA_tril", "self");
  structured_tril_cuda_functional op;
op.meta(self, diagonal);
op.impl(self, diagonal, *op.outputs_[0]);
return std::move(op.outputs_[0]).take();
}

******
m.impl("tril", TORCH_FN(wrapper_CUDA_tril));
```

该算子在实现时继承了基类 `at::native::structured_tril_cuda`，那么我们也需要这么实现：

```
#include <ATen/ops/tril_native.h>

#include "torch_musa/csrc/aten/utils/Utils.h"
#include "torch_musa/csrc/core/MUSAGuard.h"

namespace at {
namespace musa {

namespace {
struct structured_tril_musa_functional final
    : public at::native::structured_tril_cuda {
  void set_output_strided(
      int64_t output_idx,
      IntArrayRef sizes,
      IntArrayRef strides,
      TensorOptions options,
      DimnameList names) override {
    auto current_device = guard_.current_device();
    if (C10_UNLIKELY(current_device.has_value())) {
      TORCH_INTERNAL_ASSERT(
          *current_device == options.device(),
          "structured kernels don't support multi-device outputs");
```

```
    } else {
      guard_.reset_device(options.device());
    }
    outputs_[output_idx] = create_out(sizes, strides, options);
  }
  void set_output_raw_strided(
      int64_t output_idx,
      IntArrayRef sizes,
      IntArrayRef strides,
      TensorOptions options,
      DimnameList names) override {
    auto current_device = guard_.current_device();
    if (C10_UNLIKELY(current_device.has_value())) {
      TORCH_INTERNAL_ASSERT(
          *current_device == options.device(),
          "structured kernels don't support multi-device outputs");
    } else {
      guard_.reset_device(options.device());
    }
    outputs_[output_idx] = create_out(sizes, strides, options);
  }
  const Tensor& maybe_get_output(int64_t output_idx) override {
    return *outputs_[output_idx];
  }
  std::array<c10::ExclusivelyOwned<Tensor>, 1> outputs_;
  c10::musa::OptionalMUSAGuard guard_;
};
} // namespace

Tensor Tril(const Tensor& self, int64_t diagonal) {
  structured_tril_musa_functional op;
  op.meta(self, diagonal);
  op.impl(self, diagonal, *op.outputs_[0]);
  return std::move(op.outputs_[0]).take();
}
} // namespace musa
} // namespace at
```

至此，我们已经完成了通过 CUDA-Porting kernels 来适配新算子。

6.2.4 利用 CPU 实现算子

对于部分算子，如果 MUDNN 不支持，CUDA-Porting 也无法支持，可以临时中利用 CPU 后端实现该算子。主要逻辑是，先把 tensor 拷贝到 CPU 侧，在 CPU 完成计算，再将结果拷贝到 GPU 侧。可以参考下述代码：

```
Tensor& AddcDivOut(const Tensor& base, const Tensor& tensor1,
                   const Tensor& tensor2, const Scalar& value, Tensor& out) {
  auto cpu_base =
      at::empty(base.sizes(), base.options().device(DeviceType::CPU));
  auto cpu_factor1 =
      at::empty(tensor1.sizes(), tensor1.options().device(DeviceType::CPU));
  auto cpu_factor2 =
      at::empty(tensor2.sizes(), tensor2.options().device(DeviceType::CPU));
  auto cpu_out =
      at::empty(out.sizes(), out.options().device(DeviceType::CPU));
  cpu_base.copy_(base);
  cpu_factor1.copy_(tensor1);
  cpu_factor2.copy_(tensor2);
  auto result = addcdiv_out(cpu_out, cpu_base, cpu_factor1, cpu_factor2);
  out.copy_(cpu_out);
  return out;
}
```

6.2.5 添加算子单元测试

如果已经完成了新算子的适配，那么还需要添加算子单元测试，保证算子适配结果的正确性。算子测试文件在 `torch_musa/tests/unittest/operator` 目录下，参考已有算子测试添加即可，在此不展开描述。

算子测试命令如下：

```
pytest -s torch_musa/tree/main/tests/unittest/operator/xxxx.py
```

6.2.6 即将支持的特性

引入 codegen 模块，实现算子的注册代码和实现代码的生成，能进一步简化算子适配的工作量。请关注这部分工作。

7 自定义算子支持

7.1 支持自定义算子

待完善。

摩尔线程 MOORE THREADS

8 FAQ

8.1 编译安装

Q：如果在更新过 torch_musa 最新代码后，编译报错?

1. 请尝试

```
python setup.py clean
bash build.sh # 整体重新编译
```

如果还报错，可能是因为需要更新 MUSA 软件栈中某个底层软件包。

8.2 Docker 容器

Q：如果在 docker container 内部使用 torch_musa 时，报错 `ImportError: libsrv_um_MUSA.so: cannot open shared object file: No such file or directory`？

1. 请确保成功安装 mt-container-toolkit(https://mcconline.mthreads.com/software/1?id=1);
2. 在启动 docker container 时请添加 `--env MTHREADS_VISIBLE_DEVICES=all`。

8.3 适配算子

Q：如果在 CUDA-Porting 适配新算子时，编译可以通过，在 `import torch;import torch_musa` 时报错找不到符号?

```
[GCC 7.5.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> import torch_musa
Traceback (most recent call last):
  File "/opt/conda/envs/test_environment/lib/python3.8/site-packages/torch_musa-
2.0.0-py3.8-linux-x86_64.egg/torch_musa/__init__.py", line 23, in <module>
    import torch_musa._MUSAC
ImportError: /opt/conda/envs/test_environment/lib/python3.8/site-packages/torch_
musa-2.0.0-py3.8-linux-x86_64.egg/torch_musa/lib/libmusa_kernels.so: undefined s
ymbol: _ZN2at4musa3cub28exclusive_sum_in_common_typeIiiEEvPKT_PT0_l

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/opt/conda/envs/test_environment/lib/python3.8/site-packages/torch_musa-
2.0.0-py3.8-linux-x86_64.egg/torch_musa/__init__.py", line 25, in <module>
    raise ImportError("Please try running Python from a different directory!") f
rom err
ImportError: Please try running Python from a different directory!
>>> exit()
(test_environment) root@77f6215babda:/home# c++filt _ZN2at4musa3cub28exclusive_s
um_in_common_typeIiiEEvPKT_PT0_l
void at::musa::cub::exclusive_sum_in_common_type<int, int>(int const*, int*, lon
g)
(test_environment) root@77f6215babda:/home# cd -
/home/torch_musa
(test_environment) root@77f6215babda:/home/torch_musa# vim torch_musa/csrc/CMake
```

1. 先用 c++filt 查看符号名称
2. 在 PyTorch 源码中 grep 搜索这个符号：

- 如果该符号定义在 cu 文件中，那么把该 cu 文件对应的 ****.mu 文件加入到 torch_musa/csrc/CMakeList.txt 中即可。
- 如果该符号定义在 cpp 文件中，那么这是一个 bug，请向 torch_musa 提交一个 issue。
- 如果 PyTorch 中也没有这个符号，那么请在 `/usr/local/musa/` 中去 grep 搜索这个符号，这个符号可能是底层库定义的。
  - 如果找到这个符号，那么请检查是否没有链接这个底层库，查看命令可参考 `ldd path/to/site-packages/torch_musa-2.0.0-py3.8-linux-x86_64.egg/torch_musa/lib/libmusa_kernels.so`
  - 如果找到这个符号，且已经链接了对应底层库，可能底层库只暴露了这个符号，但是还未给出定义。如果是底层库暴露符号，但是 porting 的算子实际运行时没有调用符号，那么我们可以在 torch_musa 中定义一个空的实现，参考 `torch_musa/csrc/aten/ops/musa/unimplemented_functions.cpp` 。如果底层库暴露符号，porting 的算子实际运行需要调用这个符号，那么可以给底层库提交需求。

3. 如果上面都没有找到上述符号，可以在 CUDA PyTorch 环境中下 grep 搜索一下，看看 CUDA 环境中这个符号定义在哪里，再和对应 MUSA 软件模块提交需求。

8.4 问题与反馈

如果在开发或者使用 torch_musa 的过程中，遇到任何 bug 或者没支持的特性，请积极向 torch_musa (https://github.mthreads.com/mthreads/torch_musa/issues) 提交 issue，我们会及时作出反馈。提交 issue 时，请给出复现问题的代码，报错 log，并且打上对应的标签，如下面例子所示：