PR #22236 完整报告

sgl-project/sglang

[Test] Add XPU device support to unit tests

合并时间：2026-05-01 07:18

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/22236

执行摘要

- 一句话：为三个测试文件添加 XPU 设备支持
- 推荐动作：此 PR 展示了在 SGLang 中为测试添加新硬件支持的标准化方法：使用 get_device() 替代硬编码设备字符串，并相应调整跳过条件。虽然改动量小，但可以作为今后测试跨硬件适配的模板。建议阅读 test_triton_scaled_mm.py 的完整实现，以及 review 评论中关于安全调用 torch.xpu 的讨论，以避免类似问题。总体而言，值得快速浏览，但不需要深入精读。

功能与动机

为了支持 Intel XPU 设备，需要使单元测试能够无差别地在 CUDA 和 XPU 上执行。PR body 中提到 'Replace hardcoded 'cuda' device references with get_device() utility to enable tests to run on both CUDA and XPU devices.'

实现拆解

实现步骤如下：

1. 导入工具函数：在每个测试文件中添加 from sglang.srt.utils.common import get_device。
2. 设备变量化：在类的 setUp 或 setUpClass 中通过 get_device() 获取当前可用设备，并保存为实例变量或类变量。原来硬编码的 device="cuda" 全部替换为 device=self.device 或 device=cls._device。
3. 更新跳过条件：将 @unittest.skipIf(not torch.cuda.is_available(), ...) 改为 @unittest.skipIf(not (torch.cuda.is_available() or torch.xpu.is_available()), ...)，使测试在 CUDA 或 XPU 可用时均会执行（或跳过）。
4. 调整默认设备：在 TestScaledMM.setUpClass 中同步修改 torch.set_default_device 为目标设备。
5. 注意事项：test_kda_kernels.py 中第二个测试类 TestKDAGateChunkCumsum 仍使用硬编码 'cuda'，未作适配，需后续处理。

关键文件：

- test/registered/attention/test_kda_kernels.py（模块 KDA 内核测试；类别 test；类型 test-coverage；符号 TestKDAFusedSigmoidGatingRecurrent, get_device）：此文件是 KDA 内核测试，修改了第一个测试类，但第二个类未改动，体现了不完全适配，是 review

讨论的焦点。

- test/registered/quant/test_triton_scaled_mm.py（模块 缩放矩阵乘测试；类别 test；类型 test-coverage；符号 TestScaledMM, get_device）：完整展示了 setUpClass 和 _make_inputs 的设备抽象，是典型的适配模式。
- python/sglang/test/attention/test_prefix_chunk_info.py（模块 前缀块测试；类别 test；类型 test-coverage；符号 TestPrefixChunkInfo, get_device）：修改了 skip 条件和 device 赋值，展示了最简单的适配模式。

关键符号：get_device, TestKDAFusedSigmoidGatingRecurrent.setUp, TestScaledMM.setUpClass, TestScaledMM._make_inputs, TestPrefixChunkInfo.setUp

关键源码片段

test/registered/attention/test_kda_kernels.py

此文件是 KDA 内核测试，修改了第一个测试类，但第二个类未改动，体现了不完全适配，是 review 讨论的焦点。

```
import unittest
import torch
from sglang.srt.utils.common import get_device

# 注意：此文件中第二个测试类 TestKDAGateChunkCumsum 仍完全硬编码
'cuda'，本次未做任何修改。
# 对于 TestKDAFusedSigmoidGatingRecurrent，以下展示了设备适配的主要改动。

@unittest.skipIf(
    not (torch.cuda.is_available() or torch.xpu.is_available()),
    "Test requires CUDA or XPU",
)
class TestKDAFusedSigmoidGatingRecurrent(unittest.TestCase):
    def setUp(self):
        self.device = get_device() # 动态获取当前可用设备
        self.token_num = 4
        # 原来的 device="cuda" 都替换为 device=self.device
        self.query_start_loc = torch.tensor([0, 1, 2, 3, 4], device=self.device)
        self.cache_indices = torch.tensor([0, 2, 5, 8], device=self.device)
        self.local_num_heads = 8
        self.head_dim = 128
        self.cache_len = 64
        self.A_log = torch.randn(1, 1, self.local_num_heads, 1, dtype=torch.float32, device=self.
        device)
        # 其他张量类似 ...
        self.ssm_states = torch.zeros(self.cache_len, self.local_num_heads, self.head_dim, self.
        head_dim, dtype=torch.float32, device=self.device)

    def run_fused(self):
        # 方法体未改动，仅设备已由 self.device 确定
        pass
```

```
    def run_kda(self):
        # 方法体未改动
        pass
```

test/registered/quant/test_triton_scaled_mm.py

完整展示了 setUpClass 和 _make_inputs 的设备抽象，是典型的适配模式。

```
import unittest
from typing import Optional
import torch
import torch.testing
from sglang.srt.layers.quantization.fp8_kernel import triton_scaled_mm
from sglang.srt.utils.common import get_device
from sglang.test.test_utils import CustomTestCase

class TestScaledMM(CustomTestCase):
    @classmethod
    def setUpClass(cls):
        # 同时检查 CUDA 和 XPU 可用性，避免直接 torch.cuda.is_available() 抛异常
        if not (torch.cuda.is_available() or torch.xpu.is_available()):
            raise unittest.SkipTest("No CUDA or XPU device available")
        cls._device = get_device() # 获取当前可用设备
        torch.set_default_device(cls._device) # 设置默认设备为获取的设备

    def _make_inputs(self, M, K, N, in_dtype):
        # 原来 device='cuda' 全部替换为 device=self._device
        if in_dtype == torch.int8:
            a = torch.randint(-8, 8, (M, K), dtype=in_dtype, device=self._device)
            b = torch.randint(-8, 8, (K, N), dtype=in_dtype, device=self._device)
        else: # fp8
            a = torch.clamp(
                0.1 * torch.randn((M, K), dtype=torch.float16, device=self._device),
                -0.3, 0.3,
            ).to(in_dtype)
            b = torch.clamp(
                0.1 * torch.randn((K, N), dtype=torch.float16, device=self._device),
                -0.3, 0.3,
            ).to(in_dtype)
        return a, b

    def test_basic_cases(self):
        # 测试逻辑不变，内部张量已通过 self._device 创建
        pass
```

评论区精华

Review 评论主要来自 Copilot，提出了两个问题：

1. torch.xpu.is_available() 的安全调用：Copilot 指出直接使用 torch.xpu.is_available() 在没有定义 torch.xpu 的 PyTorch 环境中可能导致 AttributeError，建议用 hasattr(torch, "xpu") 或调用已有工具函数 is_xpu() 保护。此评论未得到作者或合并者回应，但 PR 仍被合并。
2. test_kda_kernels.py 的不完全适配：Copilot 发现文件中第二个测试类 TestKDAGateChunkCumsum 仍全部硬编码 device="cuda"，本次修改仅覆盖了第一个类。作者未回应或修复，PR 描述可能引起歧义。

- torch.xpu.is_available 安全调用 (correctness): 未采纳，PR 合并时未修改。
- TestKDAGateChunkCumsum 未适配 (testing): 未处理，PR 描述可能不准确。

风险与影响

- 风险：技术风险包括：
- 兼容性风险：三个文件的 skipIf 条件和 setUpClass 中直接使用 torch.xpu.is_available()，在 torch.xpu 未定义的环境中会抛出 AttributeError。目前 CI 环境可能已具备该属性，但未来扩展时需警惕。建议统一使用 is_xpu() 工具函数。
- 测试覆盖遗漏：test_kda_kernels.py 中 TestKDAGateChunkCumsum 未适配，若在 XPU CI 中运行该文件，该测试类仍会因 CUDA 不可用而跳过（因为 skipIf 未更新），不会产生错误，但会导致测试覆盖盲区。
- 回归风险低：对于纯 CUDA 环境，get_device() 返回 'cuda'，行为与原来一致，不会引入回归。
- 影响：影响范围限于三个测试文件，对用户无直接影响。对开发和 CI 团队：
- 正面影响：这三个测试可以在 XPU CI 流水线中执行或跳过，扩大测试覆盖。
- 负面影响：如果 XPU 环境配置不当，可能因 AttributeError 导致测试失败。但鉴于 PR 已合并，预计内部已验证。
- 团队影响：为后续更多测试的 XPU 适配提供了可参考的模式。
- 风险标记：安全调用 torch.xpu，部分测试未完全适配

关联脉络

- PR #23557 [Intel GPU] Integrate flash_mla_decode in Intel XPU attention backend: 同为 Intel XPU 支持，提供了注意力后端的 XPU 集成，本 PR 为相关测试提供设备适配。