PR #20672 完整报告

sgl-project/sglang

[MUSA][17/N] ci: Add MUSA diffusion, sgl-kernel tests, and CI workflow support

合并时间：2026-05-08 11:45

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/20672

执行摘要

- 一句话：为 MUSA 后端添加 CI 测试工作流与单元测试套件
- 推荐动作：该 PR 是新增硬件后端 CI 基础设施的重要贡献，值得所有关心多后端支持的开发者阅读。特别值得关注的是 run_suite_musa.py 的分区执行与失败重试设计，以及基准测试中条件导入 MUSA 核函数以兼容多后端的模式。

功能与动机

该 PR 是 #16565 中描述的为 MUSA 添加完整支持的一部分，主要目标是为 MUSA 后端建立 CI 覆盖，确保多模态扩散和 sgl-kernel 功能在 MUSA 上持续验证，及早捕获回归，同时保持代码库在 CUDA、ROCm 和 MUSA 之间的统一。

实现拆解

1. 创建 MUSA CI 工作流：在 .github/workflows/pr-test-musa.yml 中定义了支持 push、pull_request、workflow_dispatch 和 workflow_call 的工作流，包含 target_stage 和 run_all_tests 输入参数，可选择性运行特定阶段或全部 MUSA 测试。工作流触发多模态生成测试（1-GPU 和 2-GPU）以及 sgl-kernel 单元测试。
2. 添加扩散模型 MUSA 测试套件：在 python/sglang/multimodal_gen/test/run_suite_musa.py 中定义了 1-gpu-musa 和 2-gpu-musa 测试套件，并根据分区 ID 和总分片数执行并行测试采集与运行。对应的测试文件 test_server_a_musa.py、test_server_b_musa.py 和 test_server_2_gpu_a_musa.py 分别覆盖单卡图和视频扩散以及双卡场景。
3. 添加层操作单元测试：在 python/sglang/multimodal_gen/test/layers/ 下新增 test_musa_rmsnorm.py 和 test_musa_silu_and_mul.py，通过调用 forward_musa 并与 forward_native 对比，验证 MUSA 自定义算子的正确性。覆盖多种隐藏层大小、数据类型（FP16/BF16/FP32）、2D/3D 形状、残差连接、零输入和大数值等边缘情况。
4. 引入 MUSA 性能基线：python/sglang/multimodal_gen/test/server/musa/perf_baselines_musa.json 记录了扩散模型各阶段（文本编码、去噪、解码等）的期望耗时，用于性能回归门控。
5. 更新 sgl-kernel 基准测试：在 sgl-kernel/benchmark/bench_moe_topk_sigmoid.py 和 bench_moe_topk_softmax.py 中，通过条件导入 MUSA 核函数，在 calculate_diff 和基准报告中加入 MUSA 实现比较，使基准测试可在 CUDA 和 MUSA 环境下灵活运行。

同时，scripts/ci/musa/musa_install_dependency.sh 和 python/sglang/multimodal_gen/test/run_suite.py 也进行了相应调整以注册 MUSA 套件。

关键文件：

- python/sglang/multimodal_gen/test/layers/test_musa_rmsnorm.py（模块 RMSNorm 测试；类别 test；类型 test-coverage；符号 get_musa_device, TestRMSNorm, setup, _make_norm）：新增的 MUSA RMSNorm 单元测试，覆盖多种隐藏层大小、数据类型、2D/3D 输入和残差路径，是验证 MUSA 算子正确性的核心测试。
- python/sglang/multimodal_gen/test/run_suite_musa.py（模块 测试套件；类别 test；类型 test-coverage；符号 parse_args, collect_test_items, run_pytest, main）：MUSA 测试套件运行器，实现分区并行测试收集和失败重试逻辑，是 CI 执行 MUSA 测试的入口。
- python/sglang/multimodal_gen/test/layers/test_musa_silu_and_mul.py（模块 SiLU+Mul 测试；类别 test；类型 test-coverage；符号 get_musa_device, TestSiluAndMul, setup, test_forward_matches_native）：新增 MUSA SiLU+Mul 融合算子的单元测试，覆盖多种形状、数据类型、边界值和非连续输入。
- sgl-kernel/benchmark/bench_moe_topk_sigmoid.py（模块 TopK 基准；类别 source；类型 dependency-wiring；符号 musa_topk_sigmoid_fn, fn）：在已有 benchmark 中条件导入 MUSA 核函数，新增 musa_topk_sigmoid_fn 封装和基准对比行，使 benchmark 支持多后端。
- sgl-kernel/benchmark/bench_moe_topk_softmax.py（模块 TopK Softmax 基准；类别 source；类型 dependency-wiring；符号 musa_topk_softmax_fn）：与 sigmoid 版本类似，为 topk softmax 增加 MUSA 条件导入和对比函数，保持基准测试的对称性。
- .github/workflows/pr-test-musa.yml（模块 CI 工作流；类别 infra；类型 infrastructure）：新增 MUSA CI 工作流定义，是整套 CI 机制的核心，控制所有 MUSA 测试的触发和执行。
- python/sglang/multimodal_gen/test/server/musa/perf_baselines_musa.json（模块 性能基线；类别 test；类型 test-coverage）：为扩散模型在 MUSA 上的端到端性能提供基准参考，用于性能回归门控。

关键符号：get_musa_device, forward_musa, forward_native, _make_norm, run_pytest, collect_test_items, parse_args, musa_topk_sigmoid_fn, musa_topk_softmax_fn, sglang_topk_sigmoid, sglang_topk_softmax, calculate_diff

关键源码片段

python/sglang/multimodal_gen/test/layers/test_musa_rmsnorm.py

新增的 MUSA RMSNorm 单元测试，覆盖多种隐藏层大小、数据类型、2D/3D 输入和残差路径，是验证 MUSA 算子正确性的核心测试。

```
# SPDX-License-Identifier: Apache-2.0
"""
Tests for MUSA-specific RMSNorm custom op.
These tests call forward_musa directly and compare against forward_native
as the reference implementation.
"""
import pytest
```

```
import torch

# 仅在 MUSA 设备可用时运行整个模块
_musa_available = hasattr(torch, "musa") and torch.musa.is_available()
pytestmark = pytest.mark.skipif(not _musa_available, reason="MUSA device not available")

SEED = 42


def get_musa_device():
    return torch.device("musa:0")


class TestRMSNorm:
    """Tests for RMSNorm.forward_musa vs forward_native."""

    @pytest.fixture(autouse=True)
    def setup(self):
        self.device = get_musa_device()

    def _make_norm(self, hidden_size, eps=1e-6, var_hidden_size=None):
        from sglang.multimodal_gen.runtime.layers.layernorm import RMSNorm
        norm = RMSNorm(hidden_size, eps=eps, var_hidden_size=var_hidden_size)
        norm = norm.to(self.device)
        return norm

    # 无残差分支测试：多种 hidden_size 和 dtype
    @pytest.mark.parametrize("hidden_size", [64, 128, 256, 512, 1024, 2048])
    @pytest.mark.parametrize("dtype", [torch.float16, torch.bfloat16, torch.float32])
    def test_no_residual_matches_native(self, hidden_size, dtype):
        """forward_musa without residual should match forward_native."""
        norm = self._make_norm(hidden_size)
        torch.manual_seed(SEED)
        x = torch.randn(4, hidden_size, dtype=dtype, device=self.device)

        out_musa = norm.forward_musa(x.clone())
        out_native = norm.forward_native(x.clone())

        # 根据 dtype 调节容许误差
        atol = 1e-2 if dtype in (torch.float16, torch.bfloat16) else 1e-4
        rtol = 1e-2 if dtype in (torch.float16, torch.bfloat16) else 1e-4
        torch.testing.assert_close(out_musa, out_native, atol=atol, rtol=rtol)

    # 带残差分支测试
    @pytest.mark.parametrize("hidden_size", [64, 128, 256, 512, 1024])
    @pytest.mark.parametrize("dtype", [torch.float16, torch.bfloat16, torch.float32])
    def test_with_residual_matches_native(self, hidden_size, dtype):
        """forward_musa with residual should match forward_native."""
        norm = self._make_norm(hidden_size)
```

```
        torch.manual_seed(SEED)
        x = torch.randn(4, hidden_size, dtype=dtype, device=self.device)
        residual = torch.randn(4, hidden_size, dtype=dtype, device=self.device)

        # 注意：forward_musa 会就地修改输入，所以先克隆
        x_musa, res_musa = x.clone(), residual.clone()
        x_native, res_native = x.clone(), residual.clone()

        out_musa, res_out_musa = norm.forward_musa(x_musa, res_musa)
        out_native, res_out_native = norm.forward_native(x_native, res_native)

        atol = 1e-2 if dtype in (torch.float16, torch.bfloat16) else 1e-4
        rtol = 1e-2 if dtype in (torch.float16, torch.bfloat16) else 1e-4
        torch.testing.assert_close(out_musa, out_native, atol=atol, rtol=rtol)
        torch.testing.assert_close(res_out_musa, res_out_native, atol=atol, rtol=rtol)
```

python/sglang/multimodal_gen/test/run_suite_musa.py

MUSA 测试套件运行器，实现分区并行测试收集和失败重试逻辑，是 CI 执行 MUSA 测试的入口。

```
"""
Test runner for multimodal_gen MUSA suites that manages partitioned execution.
"""
import argparse
import subprocess
import sys

from sglang.multimodal_gen.runtime.utils.logging_utils import init_logger

logger = init_logger(__name__)

# 定义 MUSA 测试套件，每套包含一组测试文件
SUITES = {
    "1-gpu-musa": [
        "musa/test_server_a_musa.py",
        "musa/test_server_b_musa.py",
    ],
    "2-gpu-musa": [
        "musa/test_server_2_gpu_a_musa.py",
    ],
}


def parse_args():
    """解析命令行参数，支持套件、分区、过滤等选项"""
    parser = argparse.ArgumentParser(description="Run multimodal_gen MUSA test suite")
    parser.add_argument("--suite", type=str, required=True, choices=list(SUITES.keys()),
                        help="The test suite to run")
    parser.add_argument("--partition-id", type=int, default=0,
```

```
                        help="Index of the current partition (for parallel execution)")
    parser.add_argument("--total-partitions", type=int, default=1,
                        help="Total number of partitions")
    parser.add_argument("-k", "--filter", type=str, default=None,
                        help="Pytest filter expression (passed to pytest -k)")
    parser.add_argument("--continue-on-error", action="store_true", default=False,
                        help="Continue running remaining tests even if one fails.")
    return parser.parse_args()


def collect_test_items(files, filter_expr=None):
    """使用 pytest --collect-only 收集测试项，返回节点 ID 列表"""
    cmd = [sys.executable, "-m", "pytest", "--collect-only", "-q"]
    if filter_expr:
        cmd.extend(["-k", filter_expr])
    cmd.extend(files)

    print(f"Collecting tests with command: {' '.join(cmd)}")
    result = subprocess.run(cmd, capture_output=True, text=True)

    # exit code 5 表示无测试被收集（可能是过滤导致的），视为正常
    if result.returncode not in (0, 5):
        error_msg = (
            f"pytest --collect-only failed with exit code {result.returncode}\n"
            f"Command: {' '.join(cmd)}\n"
        )
        if result.stderr:
            error_msg += f"stderr:\n{result.stderr}\n"
        if result.stdout:
            error_msg += f"stdout:\n{result.stdout}\n"
        logger.error(error_msg)
        raise RuntimeError(error_msg)

    if result.returncode == 5:
        print("No tests were collected (exit code 5). This may be expected with filters.")

    test_items = []
    for line in result.stdout.strip().split("\n"):
        line = line.strip()
        if line and "::" in line and not line.startswith(("=", "-", " ")):
            test_id = line.split()[0] if " " in line else line
            if "::" in test_id:
                test_items.append(test_id)

    print(f"Collected {len(test_items)} test items")
    return test_items
```

评论区精华

- yeahdongcn 指出 CI 工作流构建 sgl-kernel 依赖于尚未合并的 #17946，需要在合并后 rebase 再测试。
- gemini-code-assist[bot] 建议在 test_musa_rmsnorm.py 和 test_musa_silu_and_mul.py 中使用 @pytest.mark.parametrize 替代 for 循环遍历数据类型，以提升测试可调试性。
- 作者采纳建议后，yeahdongcn 审核并批准。
- CI 构建依赖 #17946 (other): #17946 已合并，作者成功 rebase，CI 正常工作。
- 使用 pytest.mark.parametrize 以提升测试隔离性 (testing): 作者采纳建议，已修改代码。

风险与影响

- 风险：
  1. 硬件依赖性：MUSA 测试仅在配备摩尔线程 GPU 的 CI runner 上执行，其他环境自动跳过，可能掩盖跨后端通用逻辑的回归。
  2. 性能基线漂移：perf_baselines_musa.json 中的期望值基于特定硬件抓取，若 CI 池硬件变更或驱动升级，可能导致误报或漏报。
  3. 维护成本：新增 14 个文件和 CI 工作流，增加了后续维护和调试的成本，特别是非 MUSA 环境下条件导入的完整性需要持续关注。
- 影响：
  - 用户：MUSA 设备用户将受益于持续验证，降低新版本中 MUSA 后端的退化风险。
  - 系统：多模态生成和 sgl-kernel 在 MUSA 上的正确性和性能得到保障。
  - 团队：需要维护 MUSA 特定的 CI 流程和测试套件，但有助于在早期捕获跨后端问题。
  - 风险标记：硬件依赖性，性能基线漂移，维护成本增加

关联脉络

- PR #17946 sgl-kernel: support MUSA build: 本 PR 的 CI 工作流依赖该 PR 中的 sgl-kernel 构建脚本，该 PR 已合并后本 PR 进行 rebase。
- PR #16565 MUSA full support tracking: 本 PR 是该 tracking issue 中的一个步骤，为 MUSA 添加 CI 覆盖。