PR #25139 完整报告

sgl-project/sglang

Migrate Intel CPU cases to the test/registered

合并时间：2026-05-14 10:22

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25139

执行摘要

- 一句话：迁移 Intel CPU 测试用例到统一注册体系
- 推荐动作：作为基础设施标准化 PR，值得关注其设计的注册模式和共享工具类。utils.py 中的 parametrize 装饰器实现简洁实用，可复用于其他测试模块。建议精读 utils.py 和 test_qkv_proj_with_rope.py，了解如何为复杂算子构造参考实现和参数化测试。

功能与动机

PR body 明确指出目标是 'improve CPU CI coverage and refine Xeon CI triggering'。原有 CPU 测试零散存放，缺乏统一的 CI 编排和注册机制，导致覆盖率不足且难以维护。通过迁移到 test/registered 统一注册体系，可以标准化测试执行、简化 CI 配置。

实现拆解

实现分为三步：

1. 集中测试文件：在 test/registered/cpu/ 下新增 22 个测试文件，覆盖 CPU 上的关键算子路径，包括激活函数、bindings、bmm、causal_conv1d、decode/extend、flash_attn、gemm、AMX attention backend、mamba、mla、moe、norm、qkv+rope、qwen3、shared_expert、topk、rope 等。每个文件顶部调用 register_cpu_ci(est_time=10, suite='stage-b-test-cpu') 注册到目标 CI 套件。
2. 引入共享工具模块：新增 test/registered/cpu/utils.py，提供 parametrize 装饰器（基于 subTest）、精度阈值字典 precision、量化辅助函数（per_token_quant_int8、convert_weight、native_w8a8_per_token_matmul）、以及激活函数包装（SiluAndMul、GeluAndMul）。这些工具被所有测试文件统一引用，确保断言标准一致。
3. 更新 CI 编排：修改 CI 工作流（具体文件未在本次材料中列出），使 stage-b-test-cpu 阶段使用统一测试运行器执行 test/registered/cpu/ 下的注册用例，并将该阶段加入 CPU per-commit 套件，替代原有分散的测试步骤。这一步确保了迁移后的测试在 CI 流水线中正确触发。

关键文件：

- test/registered/cpu/utils.py（模块 测试工具；类别 test；类型 test-coverage；符号 parametrize, SiluAndMul, GeluAndMul, per_token_quant_int8）：所有 CPU 测试的共享基石；定义了精度阈值、参数化装饰器、量化辅助函数和参考激活函数，确保断言标准统一，是本次迁移的核心工具模块。

- test/registered/cpu/test_qkv_proj_with_rope.py（模块 算子测试；类别 test；类型 test-coverage；符号 layernorm, rotary_emb, native_torch, native_torch_int8）：覆盖 CPU 上 QKV 投影与旋转位置编码融合算子的完整测试，包含 bf16、int8、fp8 三种量化模式，参考实现详尽，是本次迁移中用例最丰富的文件之一。
- test/registered/cpu/test_moe.py（模块 算子测试；类别 test；类型 test-coverage；符号 fused_moe, TestFusedExperts, _bf16_moe, test_bf16_moe）：CPU MoE 融合专家算子的测试，覆盖 bf16、int8、fp8、int4 四种量化模式，包含 topk 路由和 fused_experts_cpu 内核调用，是 CPU 推理核心路径之一。

关键符号：fused_moe, native_torch, layernorm, rotary_emb, SiluAndMul, per_token_quant_int8, native_w8a8_per_token_matmul

关键源码片段

test/registered/cpu/utils.py

所有 CPU 测试的共享基石；定义了精度阈值、参数化装饰器、量化辅助函数和参考激活函数，确保断言标准统一，是本次迁移的核心工具模块。

```
# test/registered/cpu/utils.py ( 部分关键实现 )
import itertools
import torch
import torch.nn.functional as F

# 各数据类型的精度容忍度，供所有测试统一使用
precision = {
    torch.bfloat16: 1e-2,
    torch.float16: 1e-3,
    torch.float32: 1e-5,
}

# 简易参数化装饰器，基于 unittest.subTest 实现
# 用法：@parametrize(batch=[1,1024], dim=[96,512])
def parametrize(**params):
    def decorator(func):
        def wrapper(self):
            # 遍历所有参数组合
            for combo in itertools.product(*params.values()):
                kwargs = dict(zip(params.keys(), combo))
                with self.subTest(**kwargs):
                    func(self, **kwargs)
        return wrapper
    return decorator

# SiluAndMul 参考实现：gate_proj 后的激活
# 输入 x 的最后一维前半为 gate，后半为 up，输出 silu(gate) * up
def SiluAndMul(x: torch.Tensor) -> torch.Tensor:
    d = x.shape[-1] // 2
    return F.silu(x[..., :d]) * x[..., d:]
```

```
# per_token_quant_int8：逐 token 量化到 int8，返回量化后张量和 scale
def per_token_quant_int8(x):
    x = x.float()
    absmax = x.abs().max(dim=-1).values.clamp_min(1e-10).unsqueeze(-1)
    scale_x = absmax / 127
    x_q = torch.round(x.mul(127 / absmax)).to(torch.int8)
    return x_q, scale_x
```

test/registered/cpu/test_qkv_proj_with_rope.py

覆盖 CPU 上 QKV 投影与旋转位置编码融合算子的完整测试，包含 bf16、int8、fp8 三种量化模式，参考实现详尽，是本次迁移中用例最丰富的文件之一。

```
# test/registered/cpu/test_qkv_proj_with_rope.py（部分）
import torch
from sglang.srt.layers.rotary_embedding.utils import apply_rotary_emb
from utils import per_token_quant_int8, native_w8a8_per_token_matmul, precision

# 常量定义，与 DeepSeek-like 模型结构对齐
kv_lora_rank = 512
qk_head_dim = 192
qk_rope_head_dim = 64
num_heads = 22

# layernorm 参考实现（不带可偏置）
def layernorm(x, weight, variance_epsilon=1e-6, residual=None):
    orig_dtype = x.dtype
    x = x.to(torch.float32)
    variance = x.pow(2).mean(dim=-1, keepdim=True)
    x = x * torch.rsqrt(variance + variance_epsilon)
    return (x * weight).to(orig_dtype)

# 旋转位置编码参考实现
def rotary_emb(q_pe, k_pe, pos, cos_sin_cache):
    q_pe, k_pe = q_pe.float(), k_pe.float()
    cos_sin = cos_sin_cache[pos]
    cos, sin = cos_sin.chunk(2, dim=-1)
    q_pe = apply_rotary_emb(q_pe[..., :rotary_dim], cos, sin, False)
    k_pe = apply_rotary_emb(k_pe[..., :rotary_dim], cos, sin, False)
    return q_pe.to(orig_dtype), k_pe.to(orig_dtype)

# bf16 精度下的完整前向参考
def native_torch(q_input, hidden_states, q_a_proj_weight, ...):
    # 依次执行：A 投影 → layernorm → B 投影 → 拆分 nope/rope →
    # nope 通过 w_kc 映射 → latent cache 生成 → k/v 分离 →
    # k_pe 拼接 → rotary_emb → 组装 q_input/k_input/v_input
    ...
```

test/registered/cpu/test_moe.py

CPU MoE 融合专家算子的测试，覆盖 bf16、int8、fp8、int4 四种量化模式，包含 topk 路由和 fused_experts_cpu 内核调用，是 CPU 推理核心路径之一。

```
# test/registered/cpu/test_moe.py ( 部分 )
import torch
from sglang.srt.layers.amx_utils import CPUQuantMethod
from utils import torch_naive_fused_moe, torch_w8a8_per_column_fused_moe, precision

# 被测函数封装：先做 grouped topk，再调用 fused_experts_cpu
def fused_moe(a, w1, w2, score, topk, renormalize, prepack):
    topk_weights = torch.empty(B, topk, dtype=torch.float32)
    topk_ids = torch.empty(B, topk, dtype=torch.int32)
    topk_weights, topk_ids = kernel.grouped_topk_cpu(
        a, score, topk, renormalize, G=1, topk_group=1, 0, None, None
    )
    packed_w1 = kernel.convert_weight_packed(w1) if prepack else w1
    packed_w2 = kernel.convert_weight_packed(w2) if prepack else w2
    return kernel.fused_experts_cpu(
        a, packed_w1, packed_w2, topk_weights, topk_ids,
        inplace=True, quant_method=CPUQuantMethod.UNQUANT,
        ...)

class TestFusedExperts(CustomTestCase):
    # 各类精度下的测试参数列表
    M = [2, 114]; N = [32]; K = [32]; E = [4]; topk = [2]; renormalize = [False, True]
    M_int8 = [1, 39]; N_int8 = [128]; K_int8 = [256]; E_int8 = [8]; topk_int8 = [3]
    # … fp8, int4 类似

    def _bf16_moe(self, m, n, k, e, topk, renormalize):
        a = torch.randn((m, k), device='cpu', dtype=torch.bfloat16) / 10
        w1 = torch.randn((e, 2*n, k), device='cpu', dtype=torch.bfloat16) / 10
        w2 = torch.randn((e, k, n), device='cpu', dtype=torch.bfloat16) / 10
        score = torch.randn((m, e), device='cpu', dtype=torch.bfloat16)
        torch_output = torch_naive_fused_moe(a, w1, w2, score, topk, renormalize)
        fused_output = fused_moe(a, w1, w2, score, topk, renormalize, prepack=True)
        atol = rtol = precision[torch_output.dtype]
        torch.testing.assert_close(torch_output, fused_output, atol=atol, rtol=rtol)
```

评论区精华

核心讨论来自 reviewer mingfeima 在 Issue 评论中向 cyb70289 提出的建议：'please move arm related CIs to registered as well. Then we will remove all the old test cases under ...' — 这暗示团队计划后续将 ARM 测试也迁移到同一体系。cyb70289 表示确认（'sure'）。其他评论主要是机器人日常提示，无实质技术争议。

- ARM CI 迁移计划 (design): ARM test 维护者 cyb70289 已确认（'sure'），计划后续执行。

风险与影响

- 风险：主要风险在于测试迁移过程中可能遗漏某些边界场景，或者共享工具 utils.py 中的参考实现与旧有测试逻辑不一致。此外，新注册的测试统一使用 est_time=10，可能不准确，需要后续根据实际执行时间调整。由于是纯测试移动，不涉及生产代码，对系统稳定性的直接影响较低，但 CI 阶段若未正确配置可能导致 CPU 测试被跳过。
- 影响：对最终用户无影响；对系统维护者，测试组织更清晰、CI 触发更可控；对团队，统一了测试注册模式，降低了后续添加 CPU 测试的门槛，并为其他硬件平台（如 ARM）的迁移提供了参考。影响范围局限在测试和 CI 基础设施，程度中等。
- 风险标记：测试迁移可能遗漏边界场景，共享工具参考实现需与旧逻辑对齐，est_time 统一设置为 10 可能不准确

关联脉络

- PR #25193 ci: compute matrix partition counts from est_time: 同为 CI 基础设施优化，涉及测试注册和执行时间估算，与本 PR 的 est_time=10 设置相关。