PR #44639 完整报告

vllm-project/vllm

[CPU][Perf]Added tanh AOR for faster gelu activations.

合并时间：2026-07-01 14:24

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/44639

执行摘要

- 一句话：为 ARM CPU 添加 tanh AOR 加速 GELU 激活
- 推荐动作：本 PR 值得 ARM CPU 用户和管理者关注。设计上采用了分层架构（C++ 内核 → 向量化包装 → Python CustomOp → 平台配置），清晰隔离平台相关代码。建议后续将 gelu_tanh 测试从单一值扩展到随机张量，并考虑将 NEON 优化推广到更多激活函数。

功能与动机

CPU 推理中 GELU 激活的 tanh 近似是常见操作。PyTorch 的默认实现未针对 ARM NEON 进行优化，导致 ARM CPU 上性能不佳。利用 ARM 官方优化例程（AOR）可显著提升单次和批量推理吞吐。PR body 中给出了性能对比数据：对 bfloat16 形状 (4096,4096) 单线程加速 2.588x，八线程 2.378x。

实现拆解

实现分为以下步骤：

1. NEON tanh 内核编写：在 csrc/cpu/cpu_tanhf_neon.hpp 中基于 ARM-software/optimized-routines 的 tanhf AdvSIMD 实现，改写出 fast_tanhf_f32x4 函数，利用多项式近似计算 e^2x - 1 并最终得到 tanh(x) = (e^2x - 1) / (e^2x + 1)，支持特殊值处理（x 绝对值大于 9.01 时直接返回 ±1）。
2. 向量化类型集成：在 csrc/cpu/cpu_types_arm.hpp 中为 FP32Vec4、FP32Vec8、FP32Vec16 添加 tanh() 方法，调用 fast_tanhf_f32x4 对每个 128 位 NEON 寄存器执行 tanh 计算，保持与其他算子一致的向量化接口。
3. CPU 内核注册：在 csrc/cpu/activation.cpp 中添加 gelu_tanh 函数，使用模板 activation_kernel 调用 gelu_tanh_act 仿函数；并在 torch_bindings.cpp 中通过 TORCH_LIBRARY 注册为 gelu_tanh 操作，在 ops.h 中声明。
4. Python 自定义操作层：在 vllm/model_executor/layers/activation.py 中新增 GELUTanh 类（继承 CustomOp），__init__ 中通过 current_platform.is_cpu() 和 CpuArchEnum.ARM 判断是否可用，forward_cpu 调用 torch.ops._C.gelu_tanh，forward_native 回退到 F.gelu(x, approximate='tanh')。同时为 GeluAndMul 添加 forward_cpu 方法，在 ARM 上复用 gelu_tanh_and_mul 内核。
5. 平台配置自动启用：在 vllm/platforms/cpu.py 的 check_and_update_config 中，若架构为 ARM 且未显式禁用，则自动追加 +gelu_tanh 和 +gelu_and_mul 到 custom_ops 列表

，确保推理时选择优化内核。

6. 测试覆盖：在 tests/kernels/core/test_cpu_activation.py 中添加 test_cpu_gelu_tanh_and_mul 测试，利用多个边界值（包括 ±9.01、±12 等）验证 gelu_tanh_and_mul 输出与 PyTorch reference 的接近程度，使用默认 atol/rtol。
7. 构建系统调整：在 cmake/cpu_extension.cmake 中添加 cpu_tanhf_neon.hpp 的源文件条目确保编译。

关键文件：

- vllm/model_executor/layers/activation.py（模块 激活层；类别 source；类型 data-contract；符号 GELUTanh, init, forward_native, forward_cpu）：Python 侧核心文件：新增 GELUTanh 自定义操作类，并调整 GELU 和 GeluAndMul 的 forward_cpu 方法，是整个优化的业务入口。
- csrc/cpu/cpu_tanhf_neon.hpp（模块 NEON 内核；类别 source；类型 dependency-wiring）：新增的 NEON 实现文件，包含所有 tanh 计算的向量化内核，是性能提升的核心。
- tests/kernels/core/test_cpu_activation.py（模块 测试；类别 test；类型 test-coverage；符号 test_cpu_gelu_tanh_and_mul）：新增测试验证 gelu_tanh_and_mul 在不同输入下的正确性，覆盖关键边界值。
- csrc/cpu/cpu_types_arm.hpp（模块 向量化类型；类别 source；类型 dependency-wiring）：在 ARM 向量化类型中添加 tanh() 方法，使得上层内核可以统一调用，避免重复实现。

关键符号：fast_tanhf_f32x4, gelu_tanh, GELUTanh.forward_cpu, GELUTanh.forward_native, GELUTanh.init, FP32Vec4.tanh, FP32Vec8.tanh, FP32Vec16.tanh

关键源码片段

vllm/model_executor/layers/activation.py

Python 侧核心文件：新增 GELUTanh 自定义操作类，并调整 GELU 和 GeluAndMul 的 forward_cpu 方法，是整个优化的业务入口。

```
# --8<-- [start:gelu_tanh]
@CustomOp.register("gelu_tanh")
class GELUTanh(CustomOp):
    # --8<-- [end:gelu_tanh]

    def __init__(self):
        super().__init__()
        # 仅在 CPU 且 ARM 架构且 torch.ops._C 包含 gelu_tanh 时启用自定义内核
        if (
            current_platform.is_cpu()
            and current_platform.get_cpu_architecture() == CpuArchEnum.ARM
            and hasattr(torch.ops._C, "gelu_tanh")
        ):
            self.op = torch.ops._C.gelu_tanh
        else:
```

```
            self.op = None

    def forward_native(self, x: torch.Tensor) -> torch.Tensor:
        # PyTorch 原生实现，作为 fallback
        return F.gelu(x, approximate="tanh")

    def forward_cpu(self, x: torch.Tensor) -> torch.Tensor:
        if self.op:
            out = torch.empty_like(x)
            self.op(out, x)
            return out
        return self.forward_native(x)

    def forward_cuda(self, x: torch.Tensor) -> torch.Tensor:
        # CUDA 上使用原生实现
        return self.forward_native(x)
```

csrc/cpu/cpu_tanhf_neon.hpp

新增的 NEON 实现文件，包含所有 tanh 计算的向量化内核，是性能提升的核心。

```
// e^2x - 1 的内联计算
inline float32x4_t e2xm1f_inline(float32x4_t x, const TanhfConstants* d) {
    float32x2_t ln2 = vld1_f32(&d->ln2_hi);
    float32x4_t lane_consts = vld1q_f32(&d->c1);

    // 参数规约：f 落在 [-ln2/2, ln2/2] 范围内，i 为精确整数
    float32x4_t j = vrndaq_f32(vmulq_laneq_f32(x, lane_consts, 2));
    int32x4_t i = vcvtq_s32_f32(j);
    float32x4_t f = vaddq_f32(x, x);
    f = vfmsq_lane_f32(f, j, ln2, 0);
    f = vfmsq_lane_f32(f, j, ln2, 1);

    // 多项式近似 expm1(f) ~= f + f^2 * P(f)
    float32x4_t f2 = vmulq_f32(f, f);
    float32x4_t f4 = vmulq_f32(f2, f2);
    float32x4_t p01 = vfmaq_laneq_f32(d->c0, f, lane_consts, 0);
    float32x4_t p23 = vfmaq_laneq_f32(d->c2, f, lane_consts, 1);
    float32x4_t poly = vfmaq_f32(p01, f2, p23);
    poly = vfmaq_laneq_f32(poly, f4, lane_consts, 3);
    poly = vfmaq_f32(f, f2, poly);

    // scale = 2^i
    int32x4_t u = vaddq_s32(vshlq_n_s32(i, 23), d->exponent_bias);
    float32x4_t scale = vreinterpretq_f32_s32(u);
    return vfmaq_f32(vsubq_f32(scale, vdupq_n_f32(1.0f)), poly, scale);
}

// 完整 tanh 计算：tanh(x) = (e^2x - 1) / (e^2x + 1)
inline float32x4_t fast_tanhf_f32x4(float32x4_t x) {
```

```
    const TanhfConstants* d = ptr_barrier(&kTanhfConstants);
    float32x4_t q = e2xm1f_inline(x, d);
    // 检查是否需要特殊处理（x 绝对值 > 9.01 时置为 ±1）
    uint32x4_t special = vcagtq_f32(x, d->special_bound);
    if (any_u32(special)) {
        return special_case(x, q, special);
    }
    // 快速路径：tanh(x) = q / (q + 2)
    return vdivq_f32(q, vaddq_f32(q, d->two));
}
```

评论区精华

> fadara01: 建议将 cpu_tanhf_neon.hpp 放入专门的 arm_simd 文件夹，作者提议顺带移动 cpu_attn_neon_bfmmla.hpp，但 reviewer 拒绝了。最终文件位置保留在 csrc/cpu/ 下，未移动。
>
> fadara01: 建议删除 .cpp 文件，全部内联到 HPP。作者采纳，删除了 cpu_tanhf_neon.cpp。
>
> bigPYJ1151: 要求添加 current_platform.is_cpu() 检查，避免在 CUDA + ARM CPU 的混合平台上误用 ARM 路径。作者在 GELUTanh.__init__ 和 _get_gelu_pytorch_tanh 中添加了检查。
>
> fadara01: 指出 gelu_tanh_and_mul 测试不应仅限 ARM，因为该 op 对所有平台可用。作者移除了 skipif 条件。

- NEON 文件组织结构 (design): 建议未采纳，文件保持在 csrc/cpu/ 下。
- 内联 vs 分离编译 (design): 已解决，删除 .cpp 文件，函数均标记 inline。
- ARM 平台检查防止 CUDA 误用 (correctness): 已解决，添加了 is_cpu() 保护。
- 测试应跨平台 (testing): 已解决，移除了平台限制。

风险与影响

- 风险：
  1. 精度风险：ARM 优化 tanh 与 PyTorch 标准实现存在微小误差（测试使用宽松 atol/rtol），可能影响需要严格数值一致性的场景。
  2. 平台兼容性：条件编译依赖 current_platform.is_cpu() 和 CpuArchEnum.ARM，若未来 ARM 架构检测逻辑变化可能导致优化不生效。
  3. 非 ARM CPU 退化：代码中非 ARM CPU 会走 forward_native，不会退化。
  4. 测试覆盖不足：新增测试仅针对 gelu_tanh_and_mul，未单独测试 gelu_tanh 算子。此外，gelu_tanh_and_mul 测试使用了固定值而非随机张量，覆盖率有限。
- 影响：
  - 用户影响：ARM CPU 用户获得 1.7x-2.6x 的 GELU 加速，推理吞吐提升；其他平台用户无感知。

- 系统影响：新增约 130 行 NEON 代码和 46 行 Python 代码，不影响现有模块接口。
- 团队影响：为后续 ARM 优化（如其他激活函数、算子）提供了可复用的向量化架构和集成模式。
- 风险标记：ARM 专用优化，数值精度变化，平台条件编译，测试覆盖有限

关联脉络

- PR #42027 Add gelu_tanh_and_mul kernel: 本 PR 引入的 gelu_tanh_and_mul 操作来自 #42027，Review 中要求对比性能，且 GeluAndMul.forward_cpu 复用了该内核。