PR #29378 完整报告

sgl-project/sglang

[CPU] enable fused_sigmoid_mul on CPU device

合并时间：2026-06-29 09:38

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/29378

执行摘要

- 一句话：为 CPU 添加 fused_sigmoid_mul 内核，优化 Qwen3.5 门控
- 推荐动作：建议重点关注 CPU 融合算子的添加模式（sgl-kernel + TorchScript 注册 + 模型接入），可复用至其他类似激活融合场景。同时应关注 exp_u20 近似可能引入的精度差异，必要时增强测试覆盖更多 shape 和数值比较。

功能与动机

根据 PR body，之前 CPU 路径回退到 PyTorch eager（reshape + sigmoid + mul_），而 GPU 使用融合的 Triton 内核。为了提高 CPU 性能，需要添加 fused_sigmoid_mul_cpu 并接入 Qwen3.5 的 CPU 路径。

实现拆解

1. 实现融合内核：在 sgl-kernel/csrc/cpu/activation.cpp 中新增 fused_sigmoid_mul_kernel_impl 模板函数，使用 at::parallel_for 并行化，对每个 token 和 head 采用向量化循环（调用 load_float_vec2 和 exp_u20 近似）计算 attn_output * sigmoid(gate)，同时支持 2D/3D gate 和非连续张量。
2. 注册算子：在 sgl-kernel/csrc/cpu/torch_extension_cpu.cpp 中声明并注册 torch.ops.sgl_kernel.fused_sigmoid_mul_cpu，指定 Tensor(a!) 别名支持 inplace 操作。
3. 集成到模型：在 python/sglang/srt/models/qwen3_5.py 中，当运行时检测到 CPU 设备时，将 fused_sigmoid_mul 绑定为 torch.ops.sgl_kernel.fused_sigmoid_mul_cpu，并修改 self_attention 中的条件分支，使 CPU 路径使用融合算子而非原来的 if not (_is_npu or _is_cpu) 回退。
4. 测试重构与覆盖：重写 test/registered/cpu/test_activation.py，从 unittest 类过渡为 pytest 参数化函数，添加 test_fused_sigmoid_mul 测试，覆盖 2D/3D gate、非连续 gate 和 inplace 模式。

关键文件：

- sgl-kernel/csrc/cpu/activation.cpp（模块 CPU 内核；类别 source；类型 core-logic；符号 fused_sigmoid_mul_cpu, fused_sigmoid_mul_kernel_impl）：核心改动：新增 fused_sigmoid_mul 融合内核实现，包含向量化循环和 exp_u20 快速 sigmoid 近似。
- python/sglang/srt/models/qwen3_5.py（模块 模型层；类别 source；类型 data-contract）：模型集成：条件分支改为 CPU 路径使用 fused_sigmoid_mul 算子，而非原本的 eager 回

退。

- test/registered/cpu/test_activation.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestActivation, _assert_close, test_activation, test_fused_sigmoid_mul）：测试覆盖：重构为 pytest 参数化，新增 test_fused_sigmoid_mul 覆盖 2D/3D gate 和 inplace 模式。
- sgl-kernel/csrc/cpu/torch_extension_cpu.cpp（模块 算子注册；类别 source；类型 core-logic；符号 fused_sigmoid_mul_cpu）：算子注册：声明并注册 fused_sigmoid_mul_cpu 到 Torch 库，支持 inplace 语义。

关键符号：fused_sigmoid_mul_cpu, fused_sigmoid_mul_kernel_impl, test_fused_sigmoid_mul

关键源码片段

sgl-kernel/csrc/cpu/activation.cpp

核心改动：新增 fused_sigmoid_mul 融合内核实现，包含向量化循环和 exp_u20 快速 sigmoid 近似。

```
// sgl-kernel/csrc/cpu/activation.cpp ( 关键实现 )

template <typename scalar_t>
void fused_sigmoid_mul_kernel_impl(
    scalar_t* __restrict__ output,
    const scalar_t* __restrict__ input,
    const scalar_t* __restrict__ gate,
    int64_t num_tokens,
    int64_t dim,
    int64_t num_heads,
    int64_t head_dim,
    int64_t g_strideT,
    int64_t g_strideH) {
  using bVec = at::vec::Vectorized<scalar_t>;
  using fVec = at::vec::Vectorized<float>;
  constexpr int64_t kVecSize = bVec::size();
  const fVec one = fVec(1.f);
  at::parallel_for(0, num_tokens, 0, [&](int64_t begin, int64_t end) {
    for (int64_t i = begin; i < end; ++i) {
      const scalar_t* i_ptr = input + i * dim;
      const scalar_t* g_ptr = gate + i * g_strideT;
      scalar_t* o_ptr = output + i * dim;
      for (int64_t h = 0; h < num_heads; ++h) {
        const scalar_t* attn_ptr = i_ptr + h * head_dim;
        const scalar_t* gate_ptr = g_ptr + h * g_strideH;
        scalar_t* out_ptr = o_ptr + h * head_dim;
        int64_t d = 0;
#pragma GCC unroll 4
        for (; d <= head_dim - kVecSize; d += kVecSize) {
          auto [x_fvec0, x_fvec1] = load_float_vec2(attn_ptr + d);
```

```
          auto [g_fvec0, g_fvec1] = load_float_vec2(gate_ptr + d);
          // 使用 exp_u20 快速近似 sigmoid: x / (1 + exp(-g))
          x_fvec0 = x_fvec0 / (one + g_fvec0.neg().exp_u20());
          x_fvec1 = x_fvec1 / (one + g_fvec1.neg().exp_u20());
          convert_from_float_ext<scalar_t>(x_fvec0, x_fvec1).store(out_ptr + d);
        }
#pragma GCC unroll 4
        for (; d < head_dim; ++d) {
          float x_val = static_cast<float>(attn_ptr[d]);
          float g_val = static_cast<float>(gate_ptr[d]);
          out_ptr[d] = static_cast<scalar_t>(x_val / (1.f + std::exp(-g_val)));
        }
      }
    }
  });
}

at::Tensor fused_sigmoid_mul_cpu(at::Tensor& input, const at::Tensor& gate, bool inplace) {
  CHECK_DIM(2, input);
  const int64_t gate_dim = gate.dim();
  TORCH_CHECK(gate_dim == 2 || gate_dim == 3, "gate must be a 2D or 3D tensor");
  CHECK_CONTIGUOUS(input);
  CHECK_LAST_DIM_CONTIGUOUS_INPUT(gate);
  const auto st = input.scalar_type();
  CHECK_EQ(gate.scalar_type(), st);
  int64_t num_tokens = input.size(0);
  int64_t d = input.size(1);
  const bool is_gate_3d = gate_dim == 3;
  int64_t num_heads = is_gate_3d ? gate.size(1) : 1;
  int64_t head_dim = gate.size(-1);
  CHECK_EQ(gate.size(0), num_tokens);
  CHECK_EQ(d, num_heads * head_dim);
  int64_t g_strideT = gate.stride(0);
  int64_t g_strideH = is_gate_3d ? gate.stride(1) : 0;
  at::Tensor out = inplace ? input : at::empty_like(input);
  AT_DISPATCH_REDUCED_FLOATING_TYPES(st, "fused_sigmoid_mul", [&] {
    fused_sigmoid_mul_kernel_impl<scalar_t>(
        out.data_ptr<scalar_t>(),
        input.data_ptr<scalar_t>(),
        gate.data_ptr<scalar_t>(),
        num_tokens, d, num_heads, head_dim, g_strideT, g_strideH);
  });
  return out;
}
```

test/registered/cpu/test_activation.py

测试覆盖：重构为 pytest 参数化，新增 test_fused_sigmoid_mul 覆盖 2D/3D gate 和 inplace 模式。

```
# test/registered/cpu/test_activation.py ( 关键测试 )

import sys
import pytest
import torch
from utils import GeluAndMul, SiluAndMul, precision
from sglang.srt.server_args import ServerArgs, set_global_server_args_for_scheduler
from sglang.test.ci.ci_register import register_cpu_ci

register_cpu_ci(est_time=10, suite="base-b-test-cpu")
register_cpu_ci(est_time=10, suite="base-b-test-cpu-arm64")
torch.manual_seed(1234)

M = [128, 129, 257]
N = [22016, 22018]
DTYPES = [torch.float16, torch.bfloat16]

def _assert_close(ref_out, out):
    atol = rtol = precision[ref_out.dtype]
    torch.testing.assert_close(ref_out, out, atol=atol, rtol=rtol)

@pytest.mark.parametrize("gate_3d", [False, True])
@pytest.mark.parametrize("dtype", DTYPES)
@pytest.mark.parametrize("head_dim", [256])
@pytest.mark.parametrize("num_heads", [16])
@pytest.mark.parametrize("m", [1, 17, 128])
def test_fused_sigmoid_mul(m, num_heads, head_dim, dtype, gate_3d):
    x = torch.randn([m, num_heads * head_dim], dtype=dtype)
    if gate_3d:
        # 使用非连续的 3D gate 切片验证 strided 支持
        gate_storage = torch.randn([m, num_heads, head_dim * 2], dtype=dtype)
        gate = gate_storage[..., :head_dim]
        assert not gate.is_contiguous()
    else:
        gate = torch.randn_like(x)

    gate_ref = gate.reshape(m, -1) if gate_3d else gate
    # 非 inplace 比较
    _assert_close(
        x * torch.sigmoid(gate_ref),
        torch.ops.sgl_kernel.fused_sigmoid_mul_cpu(x, gate, False),
    )

    # inplace 模式：验证修改原始张量且结果一致
    x_inplace = x.clone()
    ref_inplace = x_inplace * torch.sigmoid(gate_ref)
    out_inplace = torch.ops.sgl_kernel.fused_sigmoid_mul_cpu(x_inplace, gate, True)
    assert out_inplace.data_ptr() == x_inplace.data_ptr()
    _assert_close(ref_inplace, x_inplace)
```

评论区精华

无 Review 讨论。

- 暂无高价值评论线程

风险与影响

- 风险：数值精度风险：内核使用 exp_u20 近似 sigmoid，可能与标准 sigmoid 存在微小差异，测试中仅验证了有限参数（head_dim=256, num_heads=16），未覆盖所有可能 shape。性能风险：并行粒度基于 token 数，当 token 很少时线程开销可能占主导，但当前 Qwen3.5 场景 token 通常较多。兼容性风险：要求 gate 最后维度连续，非最后维度连续会触发断言；若未来模型产生新的非连续布局可能失败。
- 影响：影响范围：仅影响在 CPU 上运行 Qwen3.5 模型的用户，加速注意力输出门控操作。不改变 GPU、NPU 等其他后端的路径。测试覆盖了新内核的基本正确性和 inplace 语义。
- 风险标记：新内核数值精度验证有限，缺少性能 benchmark

关联脉络

- PR #29382 [CPU] use faster exp in silu_and_mul: 修改了同一文件 sgl-kernel/csrc/cpu/activation.cpp，并使用了 exp_u20 加速，本 PR 也采用了相同的 exp_u20 近似。