PR #27630 完整报告

sgl-project/sglang

[AMD] Fuse sigmoid + mul attention output gate into single Triton kernel

合并时间：2026-06-11 17:05

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27630

执行摘要

- 一句话：融合 sigmoid 和乘法为一个 Triton kernel，降低 AMD GPU 延迟
- 推荐动作：本 PR 是一个针对 AMD GPU 的精准性能优化，内核实现简洁清晰，测试配套完整。值得关注的设计要点是：使用 _is_hip 条件编译避免影响 CUDA 路径；Triton kernel 内部提升精度至 float32 的做法。适合作为 Triton kernel 融合模式的入门参考。

功能与动机

在 AMD MI355 (gfx950) 上，HIP kernel 启动开销约为 4 us 每个 kernel。Qwen3.5 模型中的 AttentionDecoderLayer 使用两个独立的元素级 kernel：`gate = torch.sigmoid(gate)` 和 `attn_output = attn_output * gate`，每层总启动开销 8.6 us 而实际计算仅约 4.5 us。融合后每层可节省约 4.1 us，15 层 attention 共计 ~62 us。

实现拆解

1. 新增 Triton fused kernel(python/sglang/jit_kernel/triton/sigmoid_gate_mul.py)：定义 _sigmoid_gate_mul_kernel Triton JIT kernel，在单个 kernel launch 中完成 x * sigmoid(gate)，内部将输入提升至 float32 进行 sigmoid 计算以保持精度。
2. 修改 qwen3_5.py 和 qwen3_next.py 的 self_attention 方法：将原来的两行 gate = torch.sigmoid(gate); attn_output = attn_output * gate 替换为条件判断：如果 _is_hip 为 True，则调用 sigmoid_gate_mul；否则走原始路径。同时为 qwen3_next.py 添加了 is_hip 导入和 _is_hip 全局变量。
3. 新增单元测试(python/sglang/jit_kernel/tests/test_sigmoid_gate_mul.py)：包括正确性验证（多 shape、dtype）、输入不变性、输出 dtype 一致性和连续性检查。测试同时注册了 CUDA CI 和 AMD CI。
4. 将 gate 条件从 _use_aiter 改为 _is_hip：最初 PR 使用 _use_aiter 条件，经 review 指出该 kernel 不依赖 aiter 库，改为直接检查 _is_hip，并移除 _use_aiter 相关导入。

关键文件：

- python/sglang/jit_kernel/triton/sigmoid_gate_mul.py（模块 Triton 内核；类别 source；类型 core-logic；符号 _sigmoid_gate_mul_kernel, sigmoid_gate_mul）：核心新增文件。定义 Triton fused kernel，将 sigmoid 和乘法合并为单一 kernel launch，是本次性能优化的核心。

- python/sglang/jit_kernel/tests/test_sigmoid_gate_mul.py（模块 测试；类别 test；类型 test-coverage；符号 reference_sigmoid_gate_mul, test_sigmoid_gate_mul_correctness, test_sigmoid_gate_mul_does_not_modify_inputs, test_sigmoid_gate_mul_output_dtype）：配套测试文件。覆盖多 dtype、多 shape 的正确性测试、输入不变性、输出 dtype 一致性和连续性。同时注册了 CUDA 和 AMD CI，确保回归安全。
- python/sglang/srt/models/qwen3_next.py（模块 模型层；类别 source；类型 data-contract）：模型实现文件，将 attention 输出门的计算从两个 kernel 替换为融合 kernel（AMD 路径），并添加了 is_hip 导入和全局变量。
- python/sglang/srt/models/qwen3_5.py（模块 模型层；类别 source；类型 data-contract）：与 qwen3_next.py 相同的修改，为 Qwen3.5 模型启用融合 kernel。

关键符号：_sigmoid_gate_mul_kernel, sigmoid_gate_mul, reference_sigmoid_gate_mul, test_sigmoid_gate_mul_correctness, test_sigmoid_gate_mul_does_not_modify_inputs, test_sigmoid_gate_mul_output_dtype, test_sigmoid_gate_mul_contiguous_output

关键源码片段

python/sglang/jit_kernel/triton/sigmoid_gate_mul.py

核心新增文件。定义 Triton fused kernel，将 sigmoid 和乘法合并为单一 kernel launch，是本次性能优化的核心。

```
import torch
import triton
import triton.language as tl


@triton.jit
def _sigmoid_gate_mul_kernel(
    x_ptr,
    gate_ptr,
    out_ptr,
    n_elements,
    BLOCK_SIZE: tl.constexpr,
):
    # 设置 program id 和偏移量
    pid = tl.program_id(0)
    offsets = pid * BLOCK_SIZE + tl.arange(0, BLOCK_SIZE)
    mask = offsets < n_elements
    # 加载 x 和 gate，提升至 float32 保证 sigmoid 精度
    x = tl.load(x_ptr + offsets, mask=mask).to(tl.float32)
    g = tl.load(gate_ptr + offsets, mask=mask).to(tl.float32)
    # 核心计算：x * sigmoid(gate)
    out = x * tl.sigmoid(g)
    # 写回结果到原始 dtype
    tl.store(out_ptr + offsets, out.to(x_ptr.dtype.element_ty), mask=mask)

def sigmoid_gate_mul(x: torch.Tensor, gate: torch.Tensor) -> torch.Tensor:
```

```
    """Compute x * sigmoid(gate) in a single fused kernel."""
    out = torch.empty_like(x)
    n = x.numel()
    grid = lambda meta: (triton.cdiv(n, meta["BLOCK_SIZE"]),)
    _sigmoid_gate_mul_kernel[grid](x, gate, out, n, BLOCK_SIZE=1024)
    return out
```

python/sglang/jit_kernel/tests/test_sigmoid_gate_mul.py

配套测试文件。覆盖多 dtype、多 shape 的正确性测试、输入不变性、输出 dtype 一致性和连续性。同时注册了 CUDA 和 AMD CI，确保回归安全。

```
import sys

import pytest
import torch

from sglang.test.ci.ci_register import register_amd_ci, register_cuda_ci

# 注册 CI 标签：CUDA 约 4 分钟，AMD 约 4 分钟
register_cuda_ci(est_time=4, suite="base-b-kernel-unit-1-gpu-large")
register_amd_ci(est_time=4, suite="jit-kernel-unit-test-amd")

DEVICE = "cuda"


def reference_sigmoid_gate_mul(x, gate):
    # 参考实现：使用 PyTorch 的 sigmoid 和乘法
    return x * torch.sigmoid(gate)


@pytest.mark.parametrize("dtype", [torch.bfloat16, torch.float16, torch.float32])
@pytest.mark.parametrize(
    "shape",
    [
        (1, 4096),
        (4, 4096),
        (8, 4096),
        (32, 8192),
        (1, 128),
        (16, 16384),
    ],
)
def test_sigmoid_gate_mul_correctness(shape, dtype):
    # 核心正确性测试
    from sglang.jit_kernel.triton.sigmoid_gate_mul import sigmoid_gate_mul

    torch.manual_seed(42)
    x = torch.randn(shape, dtype=dtype, device=DEVICE)
    gate = torch.randn(shape, dtype=dtype, device=DEVICE)
```

```
    ref = reference_sigmoid_gate_mul(x, gate)
    out = sigmoid_gate_mul(x, gate)

    # bf16 要求更宽松的容忍度
    rtol = 1e-2 if dtype == torch.bfloat16 else 1e-3
    atol = 2e-2 if dtype == torch.bfloat16 else 1e-3
    torch.testing.assert_close(out, ref, rtol=rtol, atol=atol)


@pytest.mark.parametrize("shape", [(4, 4096), (1, 128)])
def test_sigmoid_gate_mul_does_not_modify_inputs(shape):
    # 验证融合 kernel 不会修改输入张量
    from sglang.jit_kernel.triton.sigmoid_gate_mul import sigmoid_gate_mul

    torch.manual_seed(42)
    x = torch.randn(shape, dtype=torch.bfloat16, device=DEVICE)
    gate = torch.randn(shape, dtype=torch.bfloat16, device=DEVICE)
    x_orig = x.clone()
    gate_orig = gate.clone()

    sigmoid_gate_mul(x, gate)

    torch.testing.assert_close(x, x_orig, rtol=0, atol=0)
    torch.testing.assert_close(gate, gate_orig, rtol=0, atol=0)


def test_sigmoid_gate_mul_output_dtype():
    # 验证输出 dtype 与输入一致
    from sglang.jit_kernel.triton.sigmoid_gate_mul import sigmoid_gate_mul

    for dtype in [torch.bfloat16, torch.float16, torch.float32]:
        x = torch.randn(4, 4096, dtype=dtype, device=DEVICE)
        gate = torch.randn(4, 4096, dtype=dtype, device=DEVICE)
        out = sigmoid_gate_mul(x, gate)
        assert out.dtype == dtype, f"Expected {dtype}, got {out.dtype}"


def test_sigmoid_gate_mul_contiguous_output():
    # 验证输出是连续的
    from sglang.jit_kernel.triton.sigmoid_gate_mul import sigmoid_gate_mul

    x = torch.randn(4, 4096, dtype=torch.bfloat16, device=DEVICE)
    gate = torch.randn(4, 4096, dtype=torch.bfloat16, device=DEVICE)
    out = sigmoid_gate_mul(x, gate)
    assert out.is_contiguous()


if __name__ == "__main__":
    sys.exit(pytest.main([__file__, "-v"]))
```

评论区精华

- CI 注册：Reviewer bingxche 要求为测试注册 AMD CI，作者 yichiche 随即添加 register_amd_ci。
- 条件判断改进：Reviewer sogalin 质疑 _use_aiter 的使用，指出 kernel 与 aiter 无关，建议改用 _is_hip。作者同意并修改，同时移除了 qwen3_next.py 中不再需要的 _use_aiter 和 get_bool_env_var 导入。
- CI 注册要求 (testing): 作者 yichiche 添加了 register_amd_ci，并重写了测试文件，最终版本包含两种 CI 注册。
- 条件判断改进：_use_aiter 改为 _is_hip (design): 作者同意并修改为 if _is_hip，同时移除了 qwen3_next.py 中不再需要的 _use_aiter 和 get_bool_env_var 导入。

风险与影响

- 风险：
  - 数值精度：Triton kernel 内部将 x 和 gate 提升至 float32 执行 sigmoid，再写回原始 dtype。与逐点操作的 PyTorch 路径可能存在微小差异，测试中已设置合理的 rtol/atol 容忍度。
  - CUDA 路径保持不变：融合 kernel 仅在 _is_hip 为 True 时启用，CUDA 用户不受影响，风险低。
  - 缺少梯度测试：当前测试仅验证前向正确性，未测试反向传播。若该 kernel 用于训练场景，需额外验证梯度。
  - 维护成本：新增一个小型 Triton kernel，需要配合后续可能的架构变化。
- 影响：
  - AMD GPU 用户：在启用了融合 kernel 的模型（Qwen3.5、Qwen3Next）上，每个 attention 层的 kernel 启动次数从 2 次减少为 1 次，每层节省 ~4.1 us。E2E 吞吐测试显示 +7.8% 总吞吐提升（但可能属于噪声）。
  - 系统影响：无配置变化，无 API 变动，对非 AMD 平台透明。
  - 团队影响：提供了一种减少 kernel launch 开销的可复用模式，其他类似的 gate 融合（如 afmoe.py、bailing_moe_linear.py）可在后续 PR 中采用。
  - 风险标记：数值精度差异，仅 HIP 启用，缺少梯度测试

关联脉络

- 暂无明显关联 PR