PR #26894 完整报告

sgl-project/sglang

[AMD] Fuse compress norm+rope+hadamard into single Triton kernel

合并时间：2026-06-04 05:20

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26894

执行摘要

- 一句话：融合 AMD DSV4 压缩后处理 Norm+RoPE+Hadamard 为单个 Triton 内核
- 推荐动作：建议精读，尤其是 Triton 内核融合技巧以及使用 debug_barrier 同步 warp 的处理方式，可作为 AMD 平台上内核优化的参考范例。

功能与动机

在 DSV4 压缩器的 *decode* 路径中，RMSNorm+RoPE 和 Hadamard 变换两个连续内核均操作同一 kv_compressed 张量，且完整 head_dim（512）可保存在寄存器中。融合它们可消除一次内核启动和全局内存往返，从而降低延迟。

实现拆解

1. 在 fused_compress_triton.py 中新增 _compress_norm_rope_hadamard_kernel Triton 内核，将 RMSNorm、RoPE、Walsh-Hadamard 蝴蝶变换（9 级，dim=512）合并为单个内核启动。
2. 新增 hip_compress_fused_norm_rope_hadamard_inplace Python 包装函数，负责设置内核参数并调用 fused kernel，同时传递 Hadamard 缩放因子。
3. 在 compressor.py 的 forward_compress 方法中，当 rotate=True 且为 HIP 平台时，选择新的 fused 函数替代原有的 hip_compress_fused_norm_rope_inplace 加 rotate_activation 的两次调用路径。
4. 使用 tl.debug_barrier() 在蝴蝶变换循环中同步 warp，防止竞争条件。
5. 未添加单元测试（仅依赖现有端到端精度和性能测试）。

关键文件：

- python/sglang/srt/layers/attention/dsv4/fused_compress_triton.py（模块 压缩内核；类别 source；类型 core-logic；符号 _compress_norm_rope_hadamard_kernel, hip_compress_fused_norm_rope_hadamard_inplace）：新增 Triton 融合内核，实现 Norm+RoPE+Hadamard 单次启动。
- python/sglang/srt/layers/attention/dsv4/compressor.py（模块 压缩调度；类别 source；类型 core-logic）：修改 HIP 分支控制流，根据 rotate 标志选择新融合内核。

关键符号：_compress_norm_rope_hadamard_kernel, hip_compress_fused_norm_rope_hadamard_inplace

关键源码片段

python/sglang/srt/layers/attention/dsv4/fused_compress_triton.py

新增 Triton 融合内核，实现 Norm+RoPE+Hadamard 单次启动。

```
@triton.jit
def _compress_norm_rope_hadamard_kernel(
    kv_ptr, weight_ptr, freqs_ptr, handle_ptr,
    eps, hadamard_scale,
    kv_row_stride, freqs_row_stride, plan_row_stride,
    HEAD_DIM: tl.constexpr, ROPE_DIM: tl.constexpr,
    HEAD_BLOCK: tl.constexpr, ROPE_PAIR_BLOCK: tl.constexpr,
    COMPRESS_RATIO: tl.constexpr, IS_DECODE: tl.constexpr,
    LOG2_HEAD_DIM: tl.constexpr,
):
    work_id = tl.program_id(0)
    # 根据 IS_DECODE 标志获取行索引和位置
    if IS_DECODE:
        row = work_id
        seq_len = tl.load(handle_ptr + work_id).to(tl.int32)
        position = ((seq_len - 1) // COMPRESS_RATIO) * COMPRESS_RATIO
    else:
        plan_base = handle_ptr + work_id * plan_row_stride
        row = tl.load(plan_base + 0).to(tl.int32)
        plan_position = tl.load(plan_base + 2).to(tl.int32)
        if row < 0:
            return
        position = plan_position + 1 - COMPRESS_RATIO

    base = row.to(tl.int64) * kv_row_stride
    offs = tl.arange(0, HEAD_BLOCK)
    mask = offs < HEAD_DIM
    x = tl.load(kv_ptr + base + offs, mask=mask, other=0.0).to(tl.float32)
    w = tl.load(weight_ptr + offs, mask=mask, other=0.0).to(tl.float32)
    rms_inv = tl.rsqrt(tl.sum(x * x, axis=0) / HEAD_DIM + eps)
    x_normed = x * rms_inv * w

    # RoPE 部分：加载实部和虚部，计算旋转
    rope_start: tl.constexpr = HEAD_DIM - ROPE_DIM
    pair_offs = tl.arange(0, ROPE_PAIR_BLOCK)
    pair_mask = pair_offs < (ROPE_DIM // 2)
    x_real = tl.load(kv_ptr + base + rope_start + 2 * pair_offs, mask=pair_mask, other=0.0).to(tl.
    float32)
    x_imag = tl.load(kv_ptr + base + rope_start + 2 * pair_offs + 1, mask=pair_mask, other=0.0).
    to(tl.float32)
    w_real = tl.load(weight_ptr + rope_start + 2 * pair_offs, mask=pair_mask, other=1.0).to(tl.
    float32)
    w_imag = tl.load(weight_ptr + rope_start + 2 * pair_offs + 1, mask=pair_mask, other=1.0).
    to(tl.float32)
```

```
    x_real = x_real * rms_inv * w_real
    x_imag = x_imag * rms_inv * w_imag

    freq_base = position.to(tl.int64) * freqs_row_stride
    f_real = tl.load(freqs_ptr + freq_base + 2 * pair_offs, mask=pair_mask, other=0.0)
    f_imag = tl.load(freqs_ptr + freq_base + 2 * pair_offs + 1, mask=pair_mask, other=0.0)
    out_real = x_real * f_real - x_imag * f_imag
    out_imag = x_real * f_imag + x_imag * f_real

    # 将 Norm+RoPE 结果写入全局内存（后续蝴蝶变换将原地读取）
    tl.store(kv_ptr + base + offs, x_normed, mask=mask & (offs < rope_start))
    tl.store(kv_ptr + base + rope_start + 2 * pair_offs, out_real, mask=pair_mask)
    tl.store(kv_ptr + base + rope_start + 2 * pair_offs + 1, out_imag, mask=pair_mask)

    # Walsh-Hadamard 蝴蝶变换：通过 store-reload 模式利用 L1 缓存
    # 使用 debug_barrier 确保同一行上的 warp 之间正确同步
    for stage in tl.static_range(LOG2_HEAD_DIM):
        stride = 1 << stage
        is_even = ((offs >> stage) & 1) == 0
        partner = tl.where(is_even, offs + stride, offs - stride)
        tl.debug_barrier()
        x_self = tl.load(kv_ptr + base + offs, mask=mask)
        x_partner = tl.load(kv_ptr + base + partner, mask=mask)
        result = tl.where(is_even, x_self + x_partner, x_partner - x_self)
        if stage == LOG2_HEAD_DIM - 1:
            result = result * hadamard_scale
        tl.debug_barrier()
        tl.store(kv_ptr + base + offs, result, mask=mask)
```

python/sglang/srt/layers/attention/dsv4/compressor.py

修改 HIP 分支控制流，根据 rotate 标志选择新融合内核。

```
        if rotate:
            # 使用融合了 Norm+RoPE+Hadamard 的新内核
            hip_compress_fused_norm_rope_hadamard_inplace(
                kv_compressed,
                norm.weight,
                norm_eps,
                freqs_cis_cache,
                plan,
                head_dim,
            )
        else:
            # 当 rotate=False（C128 压缩器）时，使用原来的 Norm+RoPE 内核
            hip_compress_fused_norm_rope_inplace(
                kv_compressed,
                norm.weight,
                norm_eps,
                freqs_cis_cache,
```

```
            plan,
        )
    return kv_compressed # 直接返回，不再调用 rotate_activation
```

评论区精华

Review 中 HaiShaw 要求补充准确性测试结果，作者提供了 GSM8K 精度（0.950）通过阈值（0.94）。此外，关于蝴蝶变换中的竞争条件，作者在第二个 commit 中增加了 debug_barrier 同步，这是关键设计决策。

- 要求补充准确度测试 (correctness): 作者补充了 GSM8K 准确度结果，得分 0.950 超过阈值 0.94，问题关闭。

风险与影响

- 风险：主要风险在于新内核的正确性：多 warp 共享同一行内存可能导致竞争，作者通过 debug_barrier 已处理。另外，该实现仅适用于 AMD HIP 平台，不影响 CUDA 代码路径。缺乏独立的单元测试，仅依赖端到端测试覆盖。
- 影响：直接影响：DeepSeek-V4 模型在 AMD MI355 GPU 上的 *decode* 性能提升约 0.92%（kernel 层面提升更显著）。不影响其他 GPU 平台、非 DSV4 模型、prefill 阶段或非 rotate 路径。无 API 变更或数据格式变化。
- 风险标记：仅影响 AMD HIP 路径，缺少单元测试覆盖，需保证多 warp 同步正确

关联脉络

- 暂无明显关联 PR