PR #26208 完整报告

sgl-project/sglang

[AMD] Dsv4/pr2 compressor opt

合并时间：2026-05-26 14:54

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26208

执行摘要

- 一句话：AMD DSV4 压缩与注意力 Triton 内核融合优化
- 推荐动作：建议精读 fused kernel 的设计和 autotune 策略，尤其 _should_use_fused_dual_scope 的决策逻辑和基于 workload size 的分级 dispatch。对于涉及 online softmax 的 kernel，review 中的 NaN 修复模式值得推广。

功能与动机

This PR improves DeepSeek-V4 inference performance on AMD ROCm by reducing decode/prefill hot-path overhead in compressor, indexer, and fused attention execution. It also consolidates kernel options so we can enable high-performance fused paths with clearer runtime flags while maintaining numerical correctness checks. (摘自 PR body)

实现拆解

1. 新增 fused compress 内核：在 fused_compress_triton.py 中实现 _fused_ape_pool_norm_rope_kernel，将 APE 偏置加法、online softmax pooling、RMSNorm 和 RoPE 融合为一个 Triton kernel，避免中间张量分配和多次 kernel launch。同时提供 C4 和 C128 的 decode/prefill fused 路径。
2. 优化 gather+dequant 内核：在 triton_mla_kernels_decode_dsv4.py 中引入带 batched scale loading 的 gather+dequant kernel (_gather_dequant_dsv4_kernel)，并通过 autotune 选择 BLOCK_TK。对于小 workload 使用固定配置减少 autotune 开销；对于大 workload 使用 1D fused 变体 (truly_fused_gather_dequant_fp8_dsv4) 进一步减少 launch。
3. 新增 split-K 注意力内核：在 triton_mla_kernels_decode_splitk.py 中实现 _splitk_attention_kernel，针对大 topk 将 K 维度拆分到多个 kernel 实例，降低寄存器压力，提高占用率。配合 run_splitk_attention 调度。
4. 新增 fused QK Norm：在 fused_qk_norm.py 中实现 fused_qk_norm，将 Q 和 K 的 RMSNorm 融合为一个 kernel。
5. 集成到模型层和运行时：修改 deepseek_v4.py 以支持通过环境变量（如 SGLANG_OPT_USE_FUSED_COMPRESS、SGLANG_OPT_USE_FUSED_QK_NORM_ROPE）选择 fused 路径。调整 compressor_v2.py 和 indexer.py 以路由到新的 fused 实现。

6. 验证与基准：添加手动测试 test_fused_compress_attn_hip.py 和 sgl-kernel/benchmark/bench_dsv4_norm_rope.py，确保数值正确性和性能可观测。

关键文件：

- python/sglang/srt/layers/attention/nsa/triton_decode/triton_mla_kernels_decode_dsv4.py（模块 MLA 解码；类别 source；类型 core-logic；符号 _gather_dequant_dsv4_kernel, _gather_dequant_dsv4_kernel_fixed_128, gather_dequant_fp8_dsv4, _gather_dequant_dsv4_1d_fused_kernel）：DSV4 专用的 gather+dequant 内核与稀疏注意力 decode 入口，大幅优化 FP8 KV 缓存读取和格式转换。
- python/sglang/srt/layers/attention/dsv4/fused_compress_triton.py（模块 压缩器；类别 source；类型 dependency-wiring；符号 _fused_ape_pool_norm_rope_kernel, fused_ape_pool_norm_rope, _c4_decode_kernel, _c4_prefill_compress_kernel）：融合压缩 Triton 内核，整合 APE 加、online softmax pool、RMSNorm、RoPE 为单一 kernel，减少中间 buffer 和多次 launch。
- python/sglang/srt/layers/attention/nsa/triton_decode/triton_mla_kernels_decode_common.py（模块 注意力公共；类别 source；类型 core-logic；符号 _bucket_total_tokens, _get_workload_size_category, _unified_sparse_decode_kernel, run_unified_attention）：提供统一稀疏注意力内核、chunked attention 辅助、token range 计算等共享代码，是其他 decode 内核的基础。
- python/sglang/srt/layers/attention/dsv4/compressor_v2.py（模块 压缩器；类别 source；类型 core-logic；符号 _c128_compress_decode_kernel, _c128_compress_prefill_write_kernel, _c128_compress_prefill_compress_kernel, _compress_forward_c128_triton）：在原有 HIP 压缩器基础上新增 fused decode/prefill 内核及 fallback 路径，解决 review 中的 NaN 和无效写入问题。
- python/sglang/srt/models/deepseek_v4.py（模块 模型层；类别 source；类型 data-contract；符号 _forward_prepare_multi_stream_hip）：集成 fused 路径开关，添加多流重叠支持，是模型层的主入口。

关键符号：_fused_ape_pool_norm_rope_kernel, _gather_dequant_dsv4_kernel, _unified_sparse_decode_kernel, _splitk_attention_kernel, fused_ape_pool_norm_rope, fused_gather_dequant_fp8_dsv4

关键源码片段

python/sglang/srt/layers/attention/nsa/triton_decode/triton_mla_kernels_decode_dsv4.py

DSV4 专用的 gather+dequant 内核与稀疏注意力 decode 入口，大幅优化 FP8 KV 缓存读取和格式转换。

```
# 导入必要的模块
import triton
import triton.language as tl

# DSV4 固定常量
DSV4_D_QK = 512
```

```
DSV4_D_NOPE = 448
DSV4_D_ROPE = 64
DSV4_TILE_SIZE = 64
DSV4_BYTES_PER_TOKEN_DATA = 576 # 448 nope + 128 rope
DSV4_BYTES_PER_TOKEN_SCALE = 8 # 7 scales + 1 padding

DSV4_USE_FUSED_THRESHOLD = 150000 # 使用 fused 1D kernel 的元素数上限
DSV4_USE_FIXED_KERNEL_THRESHOLD = 32768 # 使用固定 BLOCK_TK=128 的元素数上限

@triton.autotune(
    configs=[
        # 三个候选配置：BLOCK_TK 64 适合小负载，128 中等，256 大负载
        triton.Config({'BLOCK_TK': 64}, num_warps=4, num_stages=1),
        triton.Config({'BLOCK_TK': 128}, num_warps=4, num_stages=1),
        triton.Config({'BLOCK_TK': 256}, num_warps=8, num_stages=1),
    ],
    key=['total_tokens_bucket', 'topk', 'workload_size_cat'],
)
@triton.jit
def _gather_dequant_dsv4_kernel(
    KV_Cache, Indices, TopkLength, OutputKV, OutputMask,
    total_tokens, total_tokens_bucket, topk, num_blocks, block_size,
    workload_size_cat, k_offset, s_q,
    stride_kv_block, stride_idx_t, stride_idx_k, stride_out_t, stride_out_k, stride_out_d,
    stride_mask_t, stride_mask_k,
    BLOCK_TK: tl.constexpr, D_NOPE: tl.constexpr, D_ROPE: tl.constexpr,
    BYTES_PER_TOKEN_DATA: tl.constexpr, BYTES_PER_TOKEN_SCALE: tl.constexpr,
    TILE_SIZE: tl.constexpr, HAS_TOPK_LENGTH: tl.constexpr,
):
    # 每个 block 处理 BLOCK_TK 个 (token, topk) 对
    pid = tl.program_id(0)
    offs_tk = pid * BLOCK_TK + tl.arange(0, BLOCK_TK)
    mask_tk = offs_tk < total_tokens * topk

    t_idx = offs_tk // topk
    k_idx = offs_tk % topk

    # 加载索引并标记无效 (-1)
    idx_ptrs = Indices + t_idx * stride_idx_t + k_idx * stride_idx_k
    indices = tl.load(idx_ptrs, mask=mask_tk, other=-1)
    is_invalid = indices == -1

    if HAS_TOPK_LENGTH:
        batch_idx = t_idx // s_q
        topk_len = tl.load(TopkLength + batch_idx, mask=mask_tk, other=topk)
        is_invalid = is_invalid | (k_idx >= topk_len)

    # 存储输出 mask
    mask_out_ptrs = OutputMask + t_idx * stride_mask_t + (k_idx + k_offset) * stride_mask_k
```

```
    tl.store(mask_out_ptrs, is_invalid, mask=mask_tk)

    valid_mask = mask_tk & ~is_invalid
    indices_clamped = tl.maximum(indices, 0)

    # 计算 block 索引和 offset
    block_idx = indices_clamped // block_size
    offset_in_block = indices_clamped % block_size
    block_idx_64 = block_idx.to(tl.int64)
    offset_in_block_64 = offset_in_block.to(tl.int64)

    # KV cache 基址计算
    kv_block_base = KV_Cache + block_idx_64 * stride_kv_block
    nope_rope_offset = offset_in_block_64 * BYTES_PER_TOKEN_DATA
    scale_base_offset = (block_size * BYTES_PER_TOKEN_DATA + offset_in_block_64 * BYTES_
    PER_TOKEN_SCALE)

    # 加载 scale 和 FP8 数据，执行 FP8 -> BF16 反量化（略）
    # 最终输出写到 OutputKV
```

python/sglang/srt/layers/attention/nsa/triton_decode/triton_mla_kernels_decode_common.py

提供统一稀疏注意力内核、chunked attention 辅助、token range 计算等共享代码，是其他 decode 内核的基础。

```
# 为减少 autotune 重编译，将 total_tokens 规整到最近的 2 的幂

def _bucket_total_tokens(total_tokens: int) -> int:
    '''将 total_tokens 向上取整到最近 power of 2，生成稳定的 autotune key。'''
    if total_tokens <= 0:
        return 1
    n = 1
    while n < total_tokens:
        n <<= 1
    return n


def _get_workload_size_category(total_tokens: int, topk: int) -> int:
    '''根据 total_elements = total_tokens * topk 划分 4 个类别，用于 autotune key。'''
    total_elements = total_tokens * topk
    if total_elements < 10000:
        return 0
    elif total_elements < 100000:
        return 1
    elif total_elements < 1000000:
        return 2
    else:
        return 3
```

```
@triton.autotune(
    configs=[
        # 针对 CDNA4 (gfx950) 优化：BLOCK_D 固定 128，BLOCK_N 固定 256
        # BLOCK_H 变化以覆盖不同 batch size
        triton.Config({'BLOCK_H': 16, 'BLOCK_N': 256, 'BLOCK_D': 128}, num_warps=8, num_
        stages=1),
        triton.Config({'BLOCK_H': 32, 'BLOCK_N': 256, 'BLOCK_D': 128}, num_warps=8, num_
        stages=1),
        triton.Config({'BLOCK_H': 64, 'BLOCK_N': 256, 'BLOCK_D': 128}, num_warps=8, num_
        stages=1),
        triton.Config({'BLOCK_H': 128, 'BLOCK_N': 256, 'BLOCK_D': 128}, num_warps=8, num_
        stages=1),
    ],
    key=['total_tokens_bucket', 'h_q', 'total_topk', 'd_qk'],
)
@triton.jit
def _unified_sparse_decode_kernel(
    Q, KV, Mask, AttnSink, Output, LSE,
    sm_scale, total_tokens, total_tokens_bucket, h_q, total_topk, d_qk, d_v,
    stride_q_t, stride_q_h, stride_q_d,
    stride_kv_t, stride_kv_k, stride_kv_d,
    stride_mask_t, stride_mask_k,
    stride_o_t, stride_o_h, stride_o_d,
    stride_lse_t, stride_lse_h,
    HAS_ATTN_SINK: tl.constexpr,
    BLOCK_H: tl.constexpr, BLOCK_N: tl.constexpr, BLOCK_D: tl.constexpr,
):
    # 统一注意力 kernel：支持单个 KV buffer（int64 安全）
    # ... 在线 softmax 循环
    pass
```

评论区精华

Review 中 gemini-code-assist[bot] 提出了三个高优先级问题：

- Online softmax -inf 防护缺失：compressor_v2.py 中 c128 decode/prefill kernel 的 online softmax 缺少对 score_k 为 -inf 的 guard，导致 NaN 传播。已在 commit 41cd3691 中添加 exp_cur = tl.where(score_k == float('-inf'), 0.0, tl.exp(score_k - m_new)) 修复。
- 无效索引写入导致状态缓冲区污染：fallback write 路径使用 torch.where 将无效索引替换为 0，覆盖 slot 0。已改为布尔索引仅写入有效条目。
- exp_s 未被 valid mask 屏蔽导致 running_sum 错误：fused_compress_triton.py 中 c128 kernel 的 exp_s reduction 未按 valid mask 屏蔽。已添加 `exp_s = tl.where(valid[:, None], exp_s, 0.0)`。 所有问题均被解决，最终获得 HaiShaw 和 yctseng0211 批准，AMD CI 失败为已知问题。
- Online softmax -inf 防护缺失 (correctness)：已添加 `exp_cur = tl.where(score_k == float('-inf'), 0.0, tl.exp(score_k - m_new))` 修复。

- 无效索引写入导致状态缓冲区污染 (correctness): 改为使用布尔索引仅写入有效索引。
- exp_s 未被 valid mask 屏蔽导致 running_sum 错误 (correctness): 添加 exp_s = tl.where(valid[:, None], exp_s, 0.0)。

风险与影响

- 风险:
  - 数值回归：尽管修复了 NaN，fused kernel 中的浮点运算重新排序可能导致与参考实现有微小差异，但 GSM8K 精度测试（95.1% 准确率）验证了正确性。
  - 内存安全：无效索引写入问题已修复，但类似模式可能出现在其他未覆盖 kernel 中，需要持续审计。
  - 性能不确定性：autotune 阈值（如 DSV4_USE_FUSED_THRESHOLD）基于 MI355X 调优，不同 ROCm 版本或 GPU 型号可能需要重新调整。
  - 配置组合爆炸：大量环境变量开关可能导致不可预期的行为组合，需要更系统化的测试。
  - 跨平台风险：新增 HIP 专用代码通过 is_hip_runtime() 隔离，但若条件编译有遗漏可能影响 NVIDIA 路径。
- 影响:
  - 用户：AMD ROCm 用户运行 DeepSeek V4 推理将从 fused kernel 获得 decode/prefill 性能提升，预期 throughput 提升。NVIDIA 用户无影响。新环境变量需用户了解以启用优化。
  - 系统：新增约 ~8000 行 Triton kernel 代码，增加编译时间和缓存占用，但通过 Triton autotune cache 缓解部分开销。
  - 团队：维护负担增加，尤其跨平台 kernel 的同步和数值一致性。但 fusion 模式可指导未来 AMD 优化。
  - 风险标记：数值稳定性修复，内存安全修复，autotune 阈值依赖硬件，环境变量组合爆炸，HIP 条件编译

关联脉络

- PR #25391 Support DeepSeek V4 DeepEP Waterfill: 同为 DeepSeek V4 性能优化，涉及 MoE 负载均衡，与本 PR 的 compressor/attention 优化互补。