PR #30169 完整报告

sgl-project/sglang

[GDN/KDA] Fuse SM100 CuteDSL prefill state I/O into the chunk h kernel

合并时间：2026-07-16 16:40

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30169

执行摘要

- 一句话：融合 SM100 CuteDSL 预填充状态 I/O 至 h 内核
- 推荐动作：值得精读。该 PR 展示了一个典型的 kernel fusion 模式——将 PyTorch 级别的内存操作融合进 CUDA kernel 内部，显著减少 kernel launch 和中间带宽。设计上通过可选的 index 参数保持向后兼容，测试通过 bit-identical 断言确保正确性，这些做法值得推广。

功能与动机

SM100 CuteDSL chunk 预填充封装器在每层调用时执行 *eager* 池状态 I/O：*gather* 初始状态和 *scatter* 最终状态，导致两次额外内核启动和中间张量分配。PR body 指出："This per-call PyTorch overhead is exactly the class that previously dragged the (fast) CuteDSL kernels below Triton before the workspace reuse fix." 而堆栈中的 Triton chunk 路径和解码内核已经正确融合了状态 I/O，预填充是剩余的缺失环节。

实现拆解

1. 内核层修改：kda_blackwell/kernel_h.py 和 gdn_blackwell/kernel_h.py 的 kernel_h 函数新增 state_indices 参数，H0 TMA 加载和 HT 存储根据 state_indices[seq_id] 寻址，支持直接从状态池读取 / 写入。
2. Chunk Pipeline 入口适配：kda_blackwell/__init__.py 的 chunk_kda_cutedsl 和 gdn_blackwell/__init__.py 的 chunk_gated_delta_rule_cutedsl 新增可选 h0_indices/initial_state_indices 参数。当提供索引时，h0 视为状态池，内核原位读写，返回的 final_state 即为池本身；否则保持旧的密集模式，返回新的 ht。
3. Wrapper 层简化：gdn_cutedsl.py 和 kda_cutedsl.py 的 *extend* 方法删除显式的 ssm_states[indices].contiguous() *gather* 和 index_copy_ *scatter*，直接传入整个状态池和 ssm_cache_indices（int32）。padding 槽位 (-1) 仍映射到最后一个 sentinel 槽。
4. 测试验证：新增两个测试函数 test_gdn_chunk_cutedsl_pool_mode_matches_dense 和 test_kda_chunk_cutedsl_pool_mode_matches_dense，参数化 bf16/fp32，通过在随机池中散布与密集路径完全相同的初始状态，断言输出、最终状态和被修改池行逐元素相等，且未索引行保持不变。

关键文件：

- python/sglang/srt/layers/attention/linear/kernels/gdn_cutedsl.py（模块 GDN 预填充；类别 source；类型 core-logic；符号 CuteDSLGDNKernel.extend）：核心 wrapper 修改

：extend 方法从 gather/scatter 改为 pool 模式，是功能实现的直接体现。

- python/sglang/srt/layers/attention/linear/kernels/kda_cutedsl.py（模块 KDA 预填充；类别 source；类型 core-logic；符号 CuteDSLKDAKernel.extend）：KDA 对应的 wrapper 修改，与 GDN 类似，体现融合模式。
- test/registered/attention/test_gdn_prefill_cutedsl.py（模块 测试验证；类别 test；类型 test-coverage；符号 test_gdn_chunk_cutedsl_pool_mode_matches_dense）：新增池模式 bit-identical 测试，验证 GDN 融合正确性，覆盖 bf16/fp32。
- test/registered/attention/test_kda_prefill_cutedsl.py（模块 测试验证；类别 test；类型 test-coverage；符号 test_kda_chunk_cutedsl_pool_mode_matches_dense）：KDA 对应的池模式测试，确保融合正确性。
- python/sglang/kernels/ops/attention/linear/kda_blackwell/__init__.py（模块 Chunk Pipeline；类别 infra；类型 infrastructure；符号 chunk_kda_cutedsl）：Chunk pipeline 入口，实现 dense/pool 模式分支逻辑，新增 h0_indices 参数。
- python/sglang/kernels/ops/attention/linear/gdn_blackwell/__init__.py（模块 Chunk Pipeline；类别 infra；类型 infrastructure；符号 chunk_gated_delta_rule_cutedsl）：GDN 对应的 chunk pipeline 入口，新增 initial_state_indices 参数。

关键符号：CuteDSLGDNKernel.extend, CuteDSLKDAKernel.extend, chunk_kda_cutedsl, chunk_gated_delta_rule_cutedsl, test_gdn_chunk_cutedsl_pool_mode_matches_dense, test_kda_chunk_cutedsl_pool_mode_matches_dense

关键源码片段

python/sglang/srt/layers/attention/linear/kernels/gdn_cutedsl.py

核心 wrapper 修改：extend 方法从 gather/scatter 改为 pool 模式，是功能实现的直接体现。

```
# python/sglang/srt/layers/attention/linear/kernels/gdn_cutedsl.py
# 修改后的 extend 方法——将状态 gather/scatter 融合到 h kernel 内部

def extend(
    self,
    q: torch.Tensor,
    k: torch.Tensor,
    v: torch.Tensor,
    g: torch.Tensor,
    beta: torch.Tensor,
    *,
    ssm_states: torch.Tensor,
    cache_indices: torch.Tensor,
    query_start_loc: torch.Tensor,
    **kwargs,
) -> tuple:
    head_k_dim = k.shape[-1]
    self._ensure_extend_loaded(head_k_dim)

    total_seq_len = q.shape[1]
```

```
    num_v_heads = v.shape[2]
    head_v_dim = v.shape[3]

    # L2 norm Q/K outside the kernel ( 同 flashinfer path)
    q_norm = self._l2norm_fn(q[0].contiguous()).unsqueeze(0)
    k_norm = self._l2norm_fn(k[0].contiguous()).unsqueeze(0)
    v_in = v[0].contiguous().unsqueeze(0)
    g_in = g[0].to(torch.float32).unsqueeze(0)
    beta_in = beta[0].to(torch.float32).unsqueeze(0)

    cu_seqlens = query_start_loc.to(torch.int32)

    # 池状态 I/O 融合进 h kernel 的 TMA 加载 / 存储:
    # 传入整个池 + 每序列槽索引，内核直接在对应行上读取 h0/write ht,
    # 无需 gather/scatter 内核，也无 [N, Hv, V, K] 中间张量。
    # 注意将 padding (-1) 映射到最后一个 sentinel 槽。
    ssm_cache_indices = torch.where(
        cache_indices >= 0,
        cache_indices,
        ssm_states.shape[0] - 1,
    ).to(torch.int32) # 改为 int32 以匹配内核签名

    chunk_indices, chunk_offsets = self._prepare_meta_fn(
        cu_seqlens, total_seq_len, chunk_size=64
    )

    # 直接传入 ssm_states ( 池 ) 和索引，内核返回后 state 已原位写入
    output, _ = self._extend_fn(
        q=q_norm,
        k=k_norm,
        v=v_in,
        g=g_in,
        beta=beta_in,
        initial_state=ssm_states, # 不再是 gathered 副本，而是池本身
        cu_seqlens=cu_seqlens,
        chunk_indices=chunk_indices,
        chunk_offsets=chunk_offsets,
        initial_state_indices=ssm_cache_indices, # 新增参数
    )

    # Match Triton extend interface: (output, last_recurrent_state, h).
    # The kernel already wrote state back into the pool in place.
    return output, None, None
```

test/registered/attention/test_gdn_prefill_cutedsl.py

新增池模式 bit-identical 测试，验证 GDN 融合正确性，覆盖 bf16/fp32。

```
# test/registered/attention/test_gdn_prefill_cutedsl.py
# 池模式与密集模式 bit-identical 测试
```

```
@pytest.mark.parametrize("state_dtype", [torch.bfloat16, torch.float32])
def test_gdn_chunk_cutedsl_pool_mode_matches_dense(state_dtype: torch.dtype):
    """Pool mode (initial_state_indices) must reproduce the dense gather/scatter
    path bit-for-bit: same o, same final-state rows written in place at the
    indexed pool slots, and every other pool row untouched."""
    torch.manual_seed(11)
    num_seqs = 5
    seq_lens = torch.randint(1, 130, (num_seqs,), dtype=torch.int32)
    cu_seqlens = torch.zeros(num_seqs + 1, device="cuda", dtype=torch.int32)
    cu_seqlens[1:] = seq_lens.to(device="cuda").cumsum(0)
    total_tokens = int(cu_seqlens[-1].item())

    num_k_heads = 4
    num_v_heads = 8
    head_k_dim = 128
    head_v_dim = 128
    dtype = torch.bfloat16

    q = torch.randn(1, total_tokens, num_k_heads, head_k_dim, device="cuda", dtype=dtype)
    k = torch.randn_like(q)
    v = torch.randn(1, total_tokens, num_v_heads, head_v_dim, device="cuda", dtype=dtype)
    q = F.normalize(q.float(), p=2, dim=-1).to(dtype)
    k = F.normalize(k.float(), p=2, dim=-1).to(dtype)
    a = torch.randn(1, total_tokens, num_v_heads, device="cuda", dtype=dtype)
    b = torch.randn(1, total_tokens, num_v_heads, device="cuda", dtype=dtype)
    A = torch.empty(num_v_heads, device="cuda", dtype=torch.float32).uniform_(0, 16)
    A_log = torch.log(A)
    dt = torch.exp(torch.rand(num_v_heads, device="cuda", dtype=torch.float32) * (math.log(0.1)
    - math.log(0.001)) + math.log(0.001))
    dt = torch.clamp(dt, min=1e-4)
    dt_bias = dt + torch.log(-torch.expm1(-dt))
    g = -A_log.exp().view(1, 1, num_v_heads) * F.softplus(a.float() + dt_bias.view(1, 1, num_v_
    heads))
    beta = torch.sigmoid(b.float())
    h0_dense = torch.randn(num_seqs, num_v_heads, head_v_dim, head_k_dim, device="cuda",
    dtype=state_dtype) * 0.05

    # 将相同初始状态散布到更大的池（随机槽位）
    num_slots = 64
    pool = torch.randn(num_slots, num_v_heads, head_v_dim, head_k_dim, device="cuda", dtype=
    state_dtype) * 0.05
    slots = torch.randperm(num_slots, device="cuda")[:num_seqs].to(torch.int32)
    pool[slots.long()] = h0_dense
    pool_before = pool.clone()

    chunk_indices, chunk_offsets = prepare_metadata_cutedsl(cu_seqlens, total_tokens)

    # 密集模式
    o_dense, ht_dense = chunk_gated_delta_rule_cutedsl(
```

```
        q=q, k=k, v=v, g=g, beta=beta, initial_state=h0_dense.clone(),
        cu_seqlens=cu_seqlens, chunk_indices=chunk_indices, chunk_offsets=chunk_offsets,
    )
    # 池模式
    o_pool, ht_pool = chunk_gated_delta_rule_cutedsl(
        q=q, k=k, v=v, g=g, beta=beta, initial_state=pool,
        cu_seqlens=cu_seqlens, chunk_indices=chunk_indices, chunk_offsets=chunk_offsets,
        initial_state_indices=slots,
    )
    torch.cuda.synchronize()

    # 断言：相同 kernel 和数学，仅寻址不同 → 逐位相等
    assert ht_pool is pool
    assert torch.equal(o_pool, o_dense)
    assert torch.equal(pool[slots.long()], ht_dense)
    untouched = torch.ones(num_slots, dtype=torch.bool, device="cuda")
    untouched[slots.long()] = False
    assert torch.equal(pool[untouched], pool_before[untouched])
```

评论区精华

该 PR 审核流程简洁，主要 reviewer kaixih 直接批准（LGTM），未产生实质性的技术争论。自动代码审查 bot Gemini Code Assist 未提出修改意见。变更逻辑清晰、测试完备是快速通过的原因。

- 暂无高价值评论线程

风险与影响

- 风险：核心变更位于线性注意力后端的关键路径，但通过以下措施降低风险：（1）新增池模式匹配测试覆盖 bf16 和 fp32，验证逐位等价；（2）密集模式（不传索引）保留旧接口，所有现有调用者和测试不受影响；（3）padding 槽映射逻辑与解码内核一致。潜在风险包括：若状态池维度变化（如 head 数、head_dim 不匹配），内核可能静默错误；但目前通过 TMA 形状检查约束。性能提升在高并发场景显著，但在低并发时收益很小，无退化。
- 影响：对用户：多序列预填充吞吐量提升（GDN 可达 42%，KDA 可达 16%），减少 GPU 显存占用（每层 less 两个 intermediate 张量）。对系统：减少了约 2 个 CUDA kernel launch 每层每预填充步。对团队：wrapper 代码更简洁，消除了手动 gather/scatter，降低维护成本。该变更对非 SM100 硬件（如 H100）无影响，因为 CuteDSL 内核只在 Blackwell 上使用。
- 风险标记：核心内核变更，SM100 特定，bit-identical 验证覆盖

关联脉络

- 暂无明显关联 PR