PR #50697 完整报告

vllm-project/vllm

[Kernel][Inkling] Fuse shared-expert partial addition into the Lamport collective

合并时间：2026-08-05 02:41

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/50697

执行摘要

- 一句话：Inkling 共享专家 partial 加法融合进 Lamport collective，单层省 1.6us
- 推荐动作：值得精读，尤其是 lamport.py 中 publish 阶段融合加法、避免单独 kernel launch 的思路，以及 model.py 中用 tuple 扩展数据契约的做法。建议关注：缺少自动化测试、NCCL fallback 的 in-place 合并语义、kernel 中 shared_ptr 占位传参的方式。若后续要推广到其他 MoE 模型，可以借鉴 forward_partials 的拆分模式。

功能与动机

PR body 中的 before/after 图说明了问题：原先 routed MoE 与 shared sink 两条路径的 partial 需要先在 torch.add_ 中相加，再送入 Lamport RS + SConv + AG + Norm；这引入了一次额外的全局内存读写和 kernel launch。优化目标是把 shared-expert 的 partial 加法直接融合进 Lamport collective 的 publish 阶段，减少开销。

实现拆解

实现分为三步：

1. MoE 前向拆分（vllm/models/inkling/nvidia/moe.py）：将原 InklingMoE.forward 拆为 forward_partials（返回 (out, sink_out) 两个 partial）和 forward（内部调用 forward_partials 后仍做 out.add_(sink_out)，保持旧入口兼容）。这样上层可以在不丢失 sink partial 的情况下把它延后到 Lamport 路径内合并。
2. 解码层契约扩展（vllm/models/inkling/nvidia/model.py）：新增 InklingDelta 类型别名（torch.Tensor | tuple[torch.Tensor, torch.Tensor]），_sconv_add_norm 的 delta 参数接收二元组时解包出 shared_delta，并在 Lamport 路径通过 shared_tensor= 传入 rs_sconv_ag_add_norm；NCCL fallback 路径则在 reduce-scatter 前执行 delta.add_(shared_delta) 合并。InklingDecoderLayer.forward 中对 InklingMoE 调用 mlp.forward_partials，其他 MLP 仍走 mlp(mlp_in)。
3. Lamport kernel 融合（vllm/models/inkling/nvidia/ops/lamport.py）：_publish_input_kernel 新增 shared_ptr、stride_shared_t、HAS_SHARED 参数；当 HAS_SHARED 为真时，在 publish 阶段读入 shared partial，以 float32 累加到 routed partial 后再 pack 为 bf16 对，写回对端 stage。rs_sconv_ag_add_norm 增加 shared_tensor 参数并校验其 shape、dtype、device 与 channel 连续性（stride(1)==1）。

测试配套：PR 未新增单元测试文件，仅在 body 中给出 kernel microbenchmark 与 E2E benchmark 数据；这是本 PR 的主要缺口。

关键文件：

- vllm/models/inkling/nvidia/model.py（模块 模型实现；类别 source；类型 data-contract；符号 _sconv_add_norm, InklingDecoderLayer.forward, InklingModel.forward）：核心接线文件：新增 InklingDelta 类型别名，_sconv_add_norm 解包 shared_delta 并传给 Lamport 路径或 NCCL fallback，DecoderLayer 改用 forward_partials。
- vllm/models/inkling/nvidia/moe.py（模块 MoE 模块；类别 source；类型 data-contract；符号 forward, forward_partials）：MoE 前向拆分核心：新增 forward_partials 返回双 partial，forward 保持旧行为，是融合的前提。
- vllm/models/inkling/nvidia/ops/lamport.py（模块 Lamport 内核；类别 infra；类型 infrastructure；符号 _publish_input_kernel, rs_sconv_ag_add_norm）：kernel 实际上做融合的地方：publish 阶段把 shared partial 累加进 routed partial。

关键符号：InklingMoE.forward_partials, InklingMoE.forward, _sconv_add_norm, InklingDecoderLayer.forward, rs_sconv_ag_add_norm, _publish_input_kernel

关键源码片段

vllm/models/inkling/nvidia/model.py

核心接线文件：新增 InklingDelta 类型别名，_sconv_add_norm 解包 shared_delta 并传给 Lamport 路径或 NCCL fallback，DecoderLayer 改用 forward_partials。

```
def _sconv_add_norm(
    delta: InklingDelta,
    hidden: torch.Tensor,
    sconv: InklingShortConv,
    norm: InklingRMSNorm | None,
    positions: torch.Tensor,
) -> tuple[torch.Tensor | None, torch.Tensor]:
    """``h = hidden + sconv(TP-sum(delta)); y = rmsnorm(h)``。

    Lamport 路径在 publish 阶段把 shared-expert 的 partial 也加进去；
    NCCL fallback 路径则在 reduce-scatter 前用 add_ 合并。
    """
    attn_metadata = get_forward_context().attn_metadata
    m = (
        attn_metadata.get(sconv.owner.prefix)
        if isinstance(attn_metadata, dict)
        else None
    )
    cache = sconv.owner.kv_cache
    off_s, ws = sconv.owner.stream_ranges[sconv.stream_idx]
    norm_w = norm.weight if norm is not None else None
    eps = norm.variance_epsilon if norm is not None else 0.0
    # 新契约：delta 可以是 (routed_partial, shared_partial) 二元组。
```

```
    if isinstance(delta, tuple):
        delta, shared_delta = delta
    else:
        shared_delta = None

    mm = get_lamport_rs_conv(hidden.shape[-1], sconv.kernel_size)
    if mm is not None and mm.usable(delta.shape[0]) and m is not None:
        assert cache.numel() > 0
        assert isinstance(m, InklingSconvMetadata)
        # shared_tensor 会直接进 publish kernel，与 routed partial 一起做
        # float32 累加，避免单独一次 torch.add_ 的额外 kernel launch。
        return mm.rs_sconv_ag_add_norm(
            delta,
            hidden,
            sconv.weight.squeeze(1),
            norm_w,
            eps,
            cache,
            positions,
            m.block_table,
            m.seq_idx,
            m.slot_mapping,
            off_s,
            ws,
            sconv.owner.block_size,
            shared_tensor=shared_delta,
        )

    # NCCL fallback（不支持 Lamport 时）：这里直接 in-place 加 shared。
    if shared_delta is not None:
        delta.add_(shared_delta)
    shard = tensor_model_parallel_reduce_scatter(delta, dim=-1)
    shard = sconv(shard.contiguous(), positions)
    full = tensor_model_parallel_all_gather(shard, dim=-1)
    if norm is None:
        return None, hidden + full
    return add_rmsnorm(hidden, full, norm_w, eps)
```

vllm/models/inkling/nvidia/moe.py

MoE 前向拆分核心：新增 forward_partials 返回双 partial，forward 保持旧行为，是融合的前提。

```
  def forward_partials(self, x: torch.Tensor) -> tuple[torch.Tensor, torch.Tensor]:
      router_logits = self.gate.compute_logits(x)
      num_tokens = x.shape[0]
      # 一次 gate select：routed slice 留给 FusedMoE 的 routing 函数，
      # sink gammas 是权重矩阵末尾几列。
      k = self.gate.topk
      weights, ids = self.gate.select_experts(router_logits)
```

```
        self._routed_sel = (
            router_logits,
            weights[:, :k].contiguous(),
            ids[:, :k].contiguous(),
        )
        gammas = weights[:, k:]

        # routed GEMM 与 sink 链在 aux stream 上并行执行，
        # 由两个 event 汇合（decode 尺寸时）。
        out, sink_out = maybe_execute_in_parallel(
            lambda: self.experts(hidden_states=x, router_logits=router_logits),
            lambda: self.sink_experts(x, gammas),
            self._sink_events[0],
            self._sink_events[1],
            self._sink_stream
            if num_tokens <= envs.VLLM_SHARED_EXPERTS_STREAM_TOKEN_THRESHOLD
            else None,
        )
        self._routed_sel = None
        return out, sink_out


    def forward(self, x: torch.Tensor) -> torch.Tensor:
        # 兼容旧调用：直接相加得到完整 MoE 输出。
        out, sink_out = self.forward_partials(x)
        return out.add_(sink_out)
```

vllm/models/inkling/nvidia/ops/lamport.py

kernel 实际上做融合的地方：publish 阶段把 shared partial 累加进 routed partial。

```
# _publish_input_kernel 关键段落（已省略 offset/mask 计算与尾部写回逻辑）
@triton.jit
def _publish_input_kernel(
    stage_ptr,
    shared_ptr,
    peer_ptrs,
    peer_offset_u32,
    stride_stage_t,
    stride_shared_t,
    C: tl.constexpr,
    CS: tl.constexpr,
    CS_P2: tl.constexpr,
    SPLITS: tl.constexpr,
    RANK: tl.constexpr,
    WORLD: tl.constexpr,
    HAS_SHARED: tl.constexpr,
    USE_PDL: tl.constexpr,
    launch_pdl: tl.constexpr,
):
```

```
    # ...（每个 phase 负责一个 channel split，先算出 token/split/elem 下标）
    values = tl.load(
        stage_ptr + offsets,
        mask=elem_mask,
        other=0.0,
    )
    # 如果本层有 shared-expert partial，直接在 publish 阶段读入并累加，
    # 这样消费者看到的已经是两路 partial 之和，省一次额外 kernel。
    if HAS_SHARED:
        shared = tl.load(
            shared_ptr + token * stride_shared_t + owner * CS + split * CS_P2 + elem,
            mask=elem_mask,
            other=0.0,
        )
        values = (values.to(tl.float32) + shared.to(tl.float32)).to(tl.bfloat16)
    packed = _pack_bf16_pairs(values)
    # ...（按 peer_ptrs 写回对端 stage 的逻辑）
```

评论区精华

该 PR 来自 fork，claude[bot] 自动 review 被禁用（维护者可手动触发）。PR 作者邀请了 WoosukKwon 与 yewentao256 查看，维护者 Isotr0py 人工审阅后给出 APPROVED（LGTM）。除此之外没有实质性的技术讨论线程。

- Fork 自动化 review 与人工批准 (other): 未发现实质性设计讨论；PR 获得维护者批准并合入。

风险与影响

- 风险：
  1. 数据契约变更：_sconv_add_norm 的 delta 参数从 torch.Tensor 扩展为 torch.Tensor | tuple，所有调用点（主要是 InklingDecoderLayer.forward）必须按新契约传参；类型别名 InklingDelta 只覆盖了当前路径，若未来其他模块误用 tuple 会触发解包错误。
  2. NCCL fallback 的 in-place 语义：delta.add_(shared_delta) 直接修改传入的 routed partial；当前 delta 来自 experts(...) 的返回值，无其他引用，但若未来调用方复用该 tensor 则可能产生意外副作用。
  3. kernel 占位传参：shared_tensor is None 时 _publish_input_kernel 仍把 input_tensor 作为 shared_ptr 传入，仅依赖 HAS_SHARED=False 避免读取；这种占位写法较脆弱，后续维护容易踩坑。
  4. 缺少测试覆盖：Lamport 路径只在 H200/TP=8/decode 场景验证过，其他 TP 配置、非 Lamport 支持的配置（走 NCCL fallback）均无自动化测试。 - 影响：影响范围限定在 Inkling 模型 NVIDIA 实现（vllm/models/inkling/nvidia/ 下的 model.py、moe.py、ops/lamport.py）。对 TP 解码场景，单 MoE 层 kernel 耗时从 8.54us 降至 6.94us（约 18.8%），40 层 MoE 的 forward 时间约省 64us；E2E TPOT 从 5.06ms 降至 5.03ms，收益有限但无回归。拆分出的 forward_partials 为后续把 shared-expert 进一

步融合进 SConv/AG 阶段提供了结构基础。对团队来说，MoE 前向多了一个入口，API 面略增。 - 风险标记：缺少测试覆盖，数据契约变更（InklingDelta 类型扩展），NCCLfallback 路径 in-place 修改 delta，kernel 占位传参易踩坑

关联脉络

- PR #50912 [Kimi K3 Perf] option to shard the shared expert for non mega case, 16.98 GiB memory/GPU saved: 同为 MoE shared-expert 路径的优化，属于 shared expert 计算 / 显存优化演进线。
- PR #50911 [Spec Decode] Enable fused non-causal TokenSpeed MLA for DSpark: 同类 kernel 融合思路：把额外操作融合进现有 collective/kernel 以减少 launch 次数。
- PR #49558 [Bugfix][MoE] Filter packed expert weights during EP loading: 同为 MoE 相关改动，涉及 expert 权重加载，可作上下文参考。