PR #52024 完整报告

vllm-project/vllm

Revert "[Perf][ROCm] Dual-stream decode with hipgraphs"

合并时间：2026-08-13 12:38

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/52024

执行摘要

- 一句话：回滚 ROCm 双流解码，修复 DP 混合模型 CI 失败
- 推荐动作：该 PR 值得快速浏览而非精读：作为回滚操作，代码层面没有新设计，但理解其上下文很有价值——建议结合 #48223 的原始设计（在 dispatch 前提前在 aux stream 启动 shared experts 以获得真实重叠）阅读，掌握 ROCm 双流与 hipgraph 兼容性的权衡。值得关注的设计决策是「整体回滚而非局部修复」，将 CI 稳定性置于性能优化之前；后续可跟踪重新合入的 PR，重点看 hybrid models 兼容性如何解决。

功能与动机

simondanielsson 在评论中说明："The PR previously broke CI for DP with hybrid models"，并附上 Buildkite AMD CI 构建链接。即 #48223 在合入后破坏了 DP 与混合模型组合的 CI，本 PR 通过整体 revert 撤回该改动以恢复构建稳定。

实现拆解

本 PR 是纯回滚操作，共涉及 2 个源文件，+55/-55，无测试或配置配套改动。

1. 回滚 shared_experts.py 的流同步机制：删除 #48223 新增的 maybe_sync_shared_experts_stream、_run_in_aux_stream 以及 record_stream/wait_stream 模式；恢复 __init__ 中按 DBO ubatch id 创建的 _input_ready_event/_output_ready_event 事件对，恢复 maybe_forward_async 与 wait 的异步执行 API。_determine_shared_experts_order 中平台判断从 is_cuda() 恢复为 is_cuda_alike()，并重新挂上 _should_enable_stream_overlap_heuristic（ROCm 仅在 dp_size > 1 时启用多流）。
2. 回滚 moe_runner.py 的启动时机：_forward_impl 恢复为在 routed experts dispatch 之前通过 maybe_forward_async 在 aux stream 启动 shared experts；_apply_quant_method 恢复 shared_experts_overlapping 参数并在 routed 计算后调用 _shared_experts.wait() 收尾；删除 _maybe_sync_shared_experts_stream 方法。
3. 分支同步：两次 merge main（提交 1a2b6105、6a1664e1）保持 revert 分支与主干一致，避免回滚后与其他改动冲突。
4. 验证：AndreasKaratzas 连续触发三轮 Buildkite CI（#83611、#83622、#83641），shen-shanshan 对失败任务发起 retry，最终两位 reviewer 通过。

关键文件：

- vllm/model_executor/layers/fused_moe/runner/shared_experts.py（模块 共享专家；类别 source；类型 data-contract；符号 _should_enable_stream_overlap_heuristic, maybe_forward_async, maybe_sync_shared_experts_stream, _run_in_aux_stream）：核心回滚文件：移除 dual-stream 的 record_stream/wait_stream 实现，恢复事件对同步与 maybe_forward_async/wait API，并恢复 ROCm 仅 DP 场景的 _should_enable_stream_overlap_heuristic 启发式。
- vllm/model_executor/layers/fused_moe/runner/moe_runner.py（模块 MoE 执行器；类别 source；类型 data-contract；符号 _maybe_sync_shared_experts_stream, _apply_quant_method, _forward_impl）：回滚 shared experts 启动时机：_forward_impl 恢复为 dispatch 前 maybe_forward_async，_apply_quant_method 恢复 shared_experts_overlapping 参数与 wait 收尾，并删除 _maybe_sync_shared_experts_stream。

关键符号：maybe_forward_async, wait, _should_enable_stream_overlap_heuristic, _maybe_sync_shared_experts_stream, _run_in_aux_stream, _apply_quant_method, _forward_impl

关键源码片段

vllm/model_executor/layers/fused_moe/runner/shared_experts.py

核心回滚文件：移除 dual-stream 的 record_stream/wait_stream 实现，恢复事件对同步与 maybe_forward_async/wait API，并恢复 ROCm 仅 DP 场景的 _should_enable_stream_overlap_heuristic 启发式。

```
# vllm/model_executor/layers/fused_moe/runner/shared_experts.py
# 本 PR（revert #48223）恢复后的实现：
# 使用 CUDA Event 对完成 aux stream 与主流的同步，取代
# record_stream / wait_stream 模式，并恢复 ROCm 场景启发式。

    @property
    def _should_enable_stream_overlap_heuristic(self) -> bool:
        # 回滚后恢复：ROCm 上经验表明只有 DPA（DP 部署）
        # 从多流共享专家中受益，因此按 dp_size 决定是否启用。
        if not current_platform.is_rocm():
            return True
        return self._moe_config.moe_parallel_config.dp_size > 1

    def maybe_forward_async(self, shared_experts_input: torch.Tensor) -> bool:
        """在 aux stream 上异步启动共享专家，不等待其完成。

        返回 True 表示已入队；调用方随后执行 routed experts，
        最后调用 `wait` 等待共享专家结果。
        """
        if (
            self._determine_shared_experts_order(shared_experts_input)
            != SharedExpertsOrder.MULTI_STREAM_OVERLAPPED
        ):
```

```
            return False
        assert self._stream is not None
        idx = self._output_idx
        assert self._output[idx] is None
        # 记录主流当前点，aux stream 等待该事件后再执行，
        # 保证共享专家的输入依赖已就绪（例如经过 dispatch 的 hidden_states）。
        self._input_ready_event[idx].record(current_stream())
        with torch.cuda.stream(self._stream):
            self._input_ready_event[idx].wait(self._stream)
            self._output[idx] = self._layer(shared_experts_input)
            self._output_ready_event[idx].record(self._stream)
        return True

    def wait(self) -> None:
        """阻塞主流直到 `maybe_forward_async` 的输出就绪。"""
        assert self._stream is not None
        self._output_ready_event[self._output_idx].wait(current_stream())
```

vllm/model_executor/layers/fused_moe/runner/moe_runner.py

回滚 shared experts 启动时机：_forward_impl 恢复为 dispatch 前 maybe_forward_async，_apply_quant_method 恢复 shared_experts_overlapping 参数与 wait 收尾，并删除 _maybe_sync_shared_experts_stream。

```
# vllm/model_executor/layers/fused_moe/runner/moe_runner.py
# _forward_impl 回滚后恢复的流程：在 routed experts 之前
# 先异步启动 shared experts（maybe_forward_async），
# 完成重叠后再由 _apply_quant_method 通过 wait() 收尾。

    def _forward_impl(
        self,
        hidden_states: torch.Tensor,
        router_logits: torch.Tensor,
        shared_experts_input: torch.Tensor | None,
        input_ids: torch.Tensor | None = None,
    ) -> torch.Tensor | tuple[torch.Tensor, torch.Tensor]:
        # TODO(bnell): this can be removed after MK migration is complete.
        self.routed_experts._ensure_moe_quant_config_init()

        # 若启用多流重叠，须在 routed expert dispatch 之前
        # 启动共享专家（aux stream），否则两者串行执行无重叠。
        shared_experts_overlapping = False
        if self._shared_experts is not None:
            shared_experts_overlapping = self._shared_experts.maybe_forward_async(
                shared_experts_input
            )

        # gate 可在共享专家运行期间与主流重叠执行。
        if self.gate is not None:
            if self._fse_fuse_gate:
```

```
                self._maybe_fuse_gate_weights()
                router_logits = F.linear(hidden_states, self._combined_gate_weight)
            else:
                router_logits, _ = self.gate(hidden_states)

        with self._sequence_parallel_context():
            hidden_states, router_logits = self._maybe_dispatch(
                hidden_states,
                router_logits,
            )
            shared_output, hidden_states = self._apply_quant_method(
                hidden_states=hidden_states,
                router_logits=router_logits,
                shared_experts_input=shared_experts_input,
                input_ids=input_ids,
                shared_experts_overlapping=shared_experts_overlapping,
            )
            return self._maybe_combine(shared_output, hidden_states)
```

评论区精华

> simondanielsson："The PR previously broke CI for DP with hybrid models"，这是回滚的直接原因。
>
> AndreasKaratzas触发多轮/cirun，shen-shanshan发起/ciretry，验证回滚后CI状态。
>
> AndreasKaratzas 批准时评价 "LGTM Thanks :)"，两位 reviewer（AndreasKaratzas、shen-shanshan）均 APPROVED，全程无 review comments，说明回滚本身无争议。

- 原 PR 破坏 DP + 混合模型 CI (correctness): 通过整体 revert 回滚 #48223，恢复 CI 稳定。
- CI 验证与合并流程 (testing): CI 通过后合入，回滚被确认无回归。

风险与影响

- 风险：
  1. 性能回退：ROCm + DP 场景下 *decode* TPOT 约回退 3-4%（#48223 实测数据），使用 DeepSeek-V3 等 MoE 模型的 DPA 部署会有可感知的性能损失。
  2. 回归风险：回滚恢复的是 #48223 之前的成熟实现（Event 对同步），CUDA 路径行为不变，多流重叠能力整体保留，回归风险较低。
  3. 关联改动耦合：main 上若已有依赖新 API（maybe_sync_shared_experts_stream/_run_in_aux_stream）的后续改动（如 #50874 的 DP 路由回放缓冲），回滚可能导致隐含依赖不一致，需确认主分支无残留引用。
  4. 测试覆盖：本次无新增测试，回滚正确性依赖既有 fused_moe 测试与 CI，缺少针对 DP + hybrid models 的专项回归用例。
- 影响：

1. 用户影响：ROCm + DP 的 MoE 模型推理（如 DeepSeek-V3、Qwen3 MoE）decode TPOT 指标回落约 3-4%，但获得稳定的构建与运行体验；CUDA、非 DP 的 ROCm 用户无感知。
2. 系统影响：AMD CI 中 DP + hybrid models 组合恢复绿色，消除 #48223 引入的阻断问题。
3. 代码与团队影响：fused_moe/runner 内部 API 契约恢复（maybe_forward_async/wait），影响面仅限 MoE runner 内部；团队后续需重新评估如何在不破坏 hybrid 模型的前提下重新引入 dual-stream 优化。 - 风险标记：核心路径回滚，ROCm 性能回退，无新增测试覆盖，依赖 API 恢复

关联脉络

- PR #48223 [Perf][ROCm] Dual-stream decode with hipgraphs: 本 PR 正是对 #48223 的 revert，撤销其所有变更。
- PR #48111 （关联 Issue）ROCm dual-stream decode 性能问题：#48223 声称修复 #48111，本 revert 后该问题重新开放，需后续以兼容方式解决。
- PR #50874 [Bugfix][R3] Size monolithic routing replay buffer for DP: 同属 fused_moe 层 DP 相关改动，可能隐式依赖 shared experts 流同步逻辑，回滚后需确认兼容性。