

# PR #46065 完整报告

vllm-project/vllm

[ROCm][AITER] Directly Implement AITER Custom All-reduce in CudaCommunicator

合并时间: 2026-07-06 20:16

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/46065>

## 执行摘要

- 一句话: AITER 自定义 AllReduce 整合到 CudaCommunicator
- 推荐动作: 建议仔细审查此 PR, 尤其是设计决策: 统一后端、移除冗余协议、版本兼容策略。对 ROCm 平台有重要影响, 值得在 ROCm 环境下进行充分回归测试后再合入。

## 功能与动机

之前启用 AITER 融合 AR-RMSNorm 时, 存在 vLLM 原生 CustomAllreduce 和独立 AITER CustomAllreduce 两个实例, 各自管理 IPC 缓冲区并在 CUDA 图捕获时注册, 容易导致不稳定和复杂问题 (源自 #37646)。现有 AITER 的自定义 allreduce 性能已与 vLLM 持平, 且已修复同步问题, 因此统一后端更安全、更简洁。

## 实现拆解

1. 新增封装类 AiterCustomAllreduce (位于 vllm/distributed/device\_communicators/aiter\_custom\_all\_reduce.py), 提供对 AITER 底层 CustomAllreduce 的轻量级适配, 暴露 should\_custom\_ar、custom\_all\_reduce、capture、close 等方法, 并增加 supports\_dynamic\_hidden\_dim 等属性用于版本兼容判断。
2. 修改 vllm/\_aiter\_ops.py, 删除 AiterCustomAllreduceProto 协议类, 将 fused\_allreduce\_rmsnorm 实现中的 aiter\_ar.fused\_ar\_rms 调用改为通过新封装的 .aiter\_ca.custom\_fused\_ar\_rms 调用, 统一接口。
3. 修改 CudaCommunicator (vllm/distributed/device\_communicators/cuda\_communicator.py) 初始化逻辑, 根据 VLLM\_ROCM\_USE\_AITER\_CUSTOM\_AR 标志创建 aiter\_ar\_comm 实例, 并调整 vLLM 原生 CustomAllreduce 初始化条件 (仅在 aiter\_ar\_comm 未启用时创建)。
4. 修改编译融合 pass (vllm/compilation/passes/fusion/allreduce\_rms\_fusion.py) 的 RocmAiterAllReduceFusionPass, 取消对原生 CustomAllreduce 类型的依赖, 直接从 rocm\_aiter\_ops.get\_aiter\_allreduce() 获取实例, 并利用新封装的 supports\_dynamic\_hidden\_dim 和 supports\_per\_group\_quant 属性进行版本兼容判断, 去除了旧的初始化路径。
5. 新增测试文件 tests/distributed/test\_rocm\_aiter\_custom\_ar.py (精确 allreduce 正确性测试) 和 tests/v1/e2e/general/test\_rocm\_aiter\_custom\_ar.py (端到端生成对比测试), 同时修改 conftest 和 ci 配置确保新测试在 ROCm CI 中运行。

关键文件：

- vllm/distributed/device\_communicators/aiter\_custom\_all\_reduce.py（模块 分布式通信；类别 source；类型 core-logic；符号 AiterCustomAllreduce, effective\_max\_size, init, aiter\_ca）：新增的封装类，是统一后端的核心，所有 AITER allreduce 调用通过它路由。
- vllm/\_aiter\_ops.py（模块 核心接口；类别 source；类型 dependency-wiring；符号 AiterCustomAllreduceProto, capture, close, fused\_ar\_rms）：删除了旧的协议类，调整了 fused op 实现，是接口适配的关键文件。
- tests/distributed/test\_rocm\_aiter\_custom\_ar.py（模块 单元测试；类别 test；类型 test-coverage；符号 \_configure\_aiter\_custom\_ar\_env, \_assert\_aiter\_handles\_input, graph\_allreduce, eager\_allreduce）：新增单元测试覆盖 eager 和 graph 模式下的 allreduce 正确性。
- vllm/compilation/passes/fusion/allreduce\_rms\_fusion.py（模块 编译融合；类别 source；类型 core-logic；符号 del）：简化编译 pass，移除对原生 CustomAllreduce 的依赖，直接使用新封装。
- tests/v1/e2e/general/test\_rocm\_aiter\_custom\_ar.py（模块 E2E 测试；类别 test；类型 test-coverage；符号 \_run\_generation, test\_rocm\_aiter\_custom\_ar\_e2e）：新增端到端测试，验证 AITER CA 与 vLLM CA 的生成一致性。
- vllm/distributed/device\_communicators/cuda\_communicator.py（模块 通信器；类别 source；类型 dependency-wiring）：修改初始化逻辑，创建 aiter\_ar\_comm 实例，控制双后端切换。

关键符号：AiterCustomAllreduce, effective\_max\_size, custom\_all\_reduce, should\_custom\_ar, capture, close, supports\_dynamic\_hidden\_dim, build\_supports\_per\_group\_quant, is\_custom\_all\_reduce\_enabled, initialize\_aiter\_allreduce, \_rocm\_aiter\_fused\_allreduce\_rmsnorm\_impl, test\_rocm\_aiter\_custom\_allreduce, test\_rocm\_aiter\_custom\_ar\_e2e

## 关键源码片段

### vllm/distributed/device\_communicators/aiter\_custom\_all\_reduce.py

新增的封装类，是统一后端的核心，所有 AITER allreduce 调用通过它路由。

```
# vllm 封装 AITER 的 CustomAllreduce，提供统一接口
class AiterCustomAllreduce:
    # 默认 IPC buffer 大小 (8192*1024*8*2)
    MAX_SIZE: int = 8192 * 1024 * 8 * 2

    @classmethod
    def effective_max_size(cls) -> int:
        """返回最大有效输入字节大小（一半用于安全边界）"""
        return cls.MAX_SIZE // 2

    def __init__(
        self,
        group: ProcessGroup,
```

```

device: int | str | torch.device,
max_size: int | None = None,
):
    # 延迟导入 AITER 实现, 避免编译时依赖
    from aiter.dist.device_communicators.custom_all_reduce import (
        CustomAllreduce as _AiterCustomAllreduce,
    )
    if max_size is None:
        max_size = self.MAX_SIZE
    # 初始化底层 AITER 实例
    self._impl = _AiterCustomAllreduce(group, device, max_size=max_size)

@property
def aiter_ca(self):
    """暴露底层 AITER CustomAllreduce 实例"""
    return self._impl

@property
def disabled(self) -> bool:
    return self._impl.disabled

def should_custom_ar(self, inp: torch.Tensor) -> bool:
    return self._impl.should_custom_ar(inp)

def custom_all_reduce(self, inp: torch.Tensor) -> torch.Tensor | None:
    return self._impl.custom_all_reduce(inp)

def capture(self):
    """CUDA 图捕获期间的 buffer 锁定上下文"""
    return self._impl.capture()

def close(self) -> None:
    self._impl.close()

@property
def supports_dynamic_hidden_dim(self) -> bool:
    """检测 AITER 是否支持运行时 hidden_dim。
    旧版 (v0.1.12 前) 只支持固定尺寸 512/1024/2048/4096;
    新版通过 _pool 属性判断动态支持。"""
    return hasattr(self._impl, "_pool")

@staticmethod
def build_supports_per_group_quant() -> bool:
    """检测运行中的 AITER 是否暴露 per-group AR+RMS+quant 内核。
    用于决定是否注册更细粒度的融合模式。"""
    from aiter.dist.device_communicators.custom_all_reduce import (
        CustomAllreduce as _AiterCustomAllreduce,
    )
    return hasattr(_AiterCustomAllreduce, "fused_ar_rms_per_group_quant")

```

```

@property
def supports_per_group_quant(self) -> bool:
    return self.build_supports_per_group_quant()

```

## vllm/\_aiter\_ops.py

删除了旧的协议类，调整了 fused op 实现，是接口适配的关键文件。

```

def _rocm_aiter_fused_allreduce_rmsnorm_impl(
    input_: torch.Tensor,
    residual: torch.Tensor,
    weight: torch.Tensor,
    epsilon: float,
) -> tuple[torch.Tensor, torch.Tensor]:
    # 获取全局的 AITER allreduce 实例
    aiter_ar = rocm_aiter_ops.get_aiter_allreduce()
    assert aiter_ar is not None, "aiter allreduce must be initialized"
    # 通过统一封装访问底层实例
    ca = aiter_ar.aiter_ca

    total_bytes = input_.numel() * input_.element_size()
    hidden_dim = input_.shape[-1]
    token_num = input_.shape[0]
    if input_.dtype in (torch.bfloat16, torch.float16):
        pack_size = 16 // input_.element_size()
        hidden_ok = hidden_dim % pack_size == 0 and hidden_dim // pack_size <= 1024
    else:
        hidden_ok = False
    token_ok = token_num <= 80
    # 从底层实例读取 world_size 和 nvlink 状态
    world_size = ca.world_size
    full_nvlink = ca.fully_connected

    # 根据拓扑决定是否使用 1-stage 融合路径
    if world_size == 2:
        size_ok = True
    elif full_nvlink and world_size <= 4:
        size_ok = total_bytes < 256 * 1024
    elif full_nvlink and world_size <= 8:
        size_ok = total_bytes < 128 * 1024
    else:
        size_ok = False

    use_1stage = hidden_ok and token_ok and size_ok

    # 调用统一的 custom_fused_ar_rms API
    result = ca.custom_fused_ar_rms(
        input_,
        residual,
        weight,

```

```
    epsilon,  
    use_1stage=use_1stage,  
)  
assert result is not None  
return result[0], result[1]
```

## 评论区精华

核心讨论围绕环境变量命名、默认启用时机、版本检测方法、测试覆盖等：

- 环境变量从 VLLM\_ROCM\_USE\_AITER\_CUSTOM\_ALL\_REDUCE 简化为 VLLM\_ROCM\_USE\_AITER\_CUSTOM\_AR。
- 默认启用需更多 benchmark 数据支撑，最终默认启用。
- 版本检测通过检查 `_pool` 属性判断是否支持动态 `hidden_dim`，虽被认为脆弱但暂无更好方案。
- 审查者要求添加 E2E 测试，作者新增了相关测试覆盖。
- 环境变量名称简化 (design): 采纳建议，修改为简写名。
- 默认启用决策 (performance): 默认启用（受 VLLM\_ROCM\_USE\_AITER 父开关控制）。
- 版本检测方法可靠性 (correctness): 保留当前方法，待未来 AITER 固定版本后再改进。
- 需要 E2E 测试 (testing): BadrBasowid 新增 `tests/v1/e2e/general/test_rocm_aiter_custom_ar.py`，包含快速缩减开关和 CUDA Graph 组合的生成对比测试。
- TP 跳过条件合理性 (question): BadrBasowid 解释 CI 使用 2 GPU，不会遇到 TP=1，但添加 `guard` 以防本地运行。后改为 `multi_gpu_test` 标记，明确要求 2+ GPU。

## 风险与影响

- 风险：主要风险包括：
  - 核心 `allreduce` 路径替换可能导致 CUDA 图捕获或融合 `op` 行为异常。
  - 版本兼容性：旧版 AITER 可能缺少 `custom_fused_ar_rms` 或动态 `hidden_dim` 支持，代码通过属性检测降级，但仍需谨慎评估。
  - 精度回归：修复了 Kimi-K2.5 的同步问题，但可能在其他模型上仍有隐藏的同步缺陷。
  - 由于涉及编译 `pass` 和 `device communicator`，回归影响面广。
  - 影响：对用户：ROCm 用户默认使用 AITER `custom allreduce`，简化配置，可能获得一致或更好的性能；不再需要同时维护两个后端。对系统：减少一个 IPC 缓冲区实例，降低资源占用。对团队：统一 `allreduce` 后端，后续 `fused op` 开发只需维护 AITER 路径，降低维护成本。
- 风险标记：核心路径变更，版本兼容依赖，精度回归潜在风险，CUDA 图捕获影响

## 关联脉络

- PR #37646 [Referenced] Original AITER Custom AllReduce wrapper: 本 PR 描述中提及，之前引入的 AITER CustomAllreduce Proto 方法复杂且不稳定，本 PR 在此基础上改进了实现，统一后端。