PR #50654 完整报告

vllm-project/vllm

[ROCm][Perf] Kimi-K3 Fused kernel for KDA decode

合并时间：2026-08-12 11:39

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/50654

执行摘要

- 一句话：ROCm KDA 解码融合内核，e2e 吞吐提升约 7%
- 推荐动作：值得精读。该 PR 展示了 ROCm 上把三段解码链融合为单内核的完整工程实践：从 HIP kernel 的 DPP 归约与非时序访存、CMake 的 arch 过滤构建，到 Python 侧双重门控、加载期权重布局 staging，再到面向 CUDA-graph 的 benchmark 设计。特别推荐关注 benchmarks/kernels/benchmark_kimi_k3_kda_decode.py 中关于“eager 计时会夸大融合收益、graph replay 才是决策指标”的方法论，以及测试中对 NULL_BLOCK_ID 填充行、槽位隔离等边界条件的处理。

功能与动机

PR body 说明：Kimi 的 KDA 层在每步 decode 中要对每个序列、每个 value head 依次做 causal conv1d 状态推进、gated delta-rule 递推和门控 RMSNorm，而 ROCm 上目前是三个独立 Triton kernel 依次启动。本 PR 的目标是镜像 CUDA 侧的融合实现，提供 HIP 版本的单 token decode 融合内核，以消除多次 launch 与中间张量往返；同时声明该内核目前只在 gfx950 上测试过，因此构建与派发都以 gfx950 为门槛。e2e 基准显示输出吞吐提升约 7%。

实现拆解

1. 新增 HIP 融合内核：csrc/libtorch_stable/kimi_k3/fused_kda_decode_kernel_rocm.cu（约 800 行）实现 kda_decode_fusion_kernel，按 (sequence, value head) 组织线程块。递推 state 为 [128, 128] fp32、每 token 读写一次（约 128 KiB HBM 流量）是主要瓶颈，因此采用 16-lane DPP 归约（row_reduce / block_reduce_sum3）、__builtin_nontemporal_load/store 非时序访存与提前预取整片 state 的流水设计，目标是打满 HBM 带宽而非压缩 FLOP。模板参数覆盖是否启用输出 norm、固定 head 数、是否更新 conv state、是否使用 safe gate 等组合；并显式处理 CUDA-graph 批尾的 NULL_BLOCK_ID（slot 0）填充行：输出清零、跳过 state 读写。
2. 构建接线：CMakeLists.txt 在 HIP 分支新增 FUSED_KDA_DECODE_HIP_ARCHS，用 list(FILTER ... INCLUDE REGEX "gfx950") 只在 arch 列表含 gfx950 时把该 .cu 加入 VLLM_STABLE_EXT_SRC，并通过 VLLM_ENABLE_FUSED_KDA_DECODE=1 编译定义暴露 torch.ops._C.fused_kda_decode。注释明确：多 arch 构建只要包含 gfx950 就会为列表内所有 arch 编译该源文件，运行时由 Python 侧门控拦截非 gfx950 设备。
3. 新增 Python 门控与权重加载器：vllm/models/kimi_k3/amd/ops/kda_decode.py 提供 is_fused_kda_decode_supported()，校验 head 数 ∈ (12, 24, 48, 96)、head_dim=128、

conv 宽度 =4、无 spec 解码、bf16 输入 / 状态、DS 状态布局以及 on_gfx950()，任一不满足即返回 False。同时实现 make_decode_conv1d_weight_loader() 与 make_decode_norm_weight_loader()：前者在加载期把 conv 权重镜像为内核需要的 width-major fp32 布局 [3, width, dim]，后者把门控 norm 权重 upcast 到 fp32，避免 decode 热点上的运行时转换。

4. 集成到 KDA 层：vllm/models/kimi_k3/amd/kda.py 的 KimiK3DeltaAttention.__init__ 按支持性决定是否注册 decode_conv1d_weight / decode_norm_weight 两个非持久 buffer 并替换对应 weight_loader；forward 在满足“无 spec mask、无 prefill、有 decode”时直接调用 ops.fused_kda_decode() 写 core_attn_out 后提前返回，其余路径（prefill、spec 解码、非 gfx950）走原 Triton 链。
5. 测试与基准配套：新增 tests/models/kimi_k3/test_amd_kda_decode.py，全文件以 gfx950 为 skip 条件，用 Triton 三段链作为数值参照对比融合内核输出、conv state 与递推 state（conv state 要求逐位一致，输出因 fp32 保持 vs BF16 中间舍入采用 3e-2 容差），并覆盖“不动未触及 slot”“无 norm 模式”“NULL_BLOCK 填充行输出清零且 slot 0 不被写”。新增 benchmarks/kernels/benchmark_kimi_k3_kda_decode.py，同时支持 eager 与 CUDA-graph 计时（graph 模式按 69 层单层 dispatch 均摊真实的 HIP graph 启动开销），报告 per-step 节省与 state 带宽。CI 通过 /ci run 在 Buildkite 上触发。

关键文件：

- csrc/libtorch_stable/kimi_k3/fused_kda_decode_kernel_rocm.cu（模块 融合内核；类别 source；类型 core-logic；符号 kda_decode_fusion_kernel, block_reduce_sum3, row_reduce, bf16_load）：新增 800 行 HIP 融合内核，是本次性能提升的核心；实现 conv1d update、gated delta-rule 递推与门控 RMSNorm 的单 launch 融合，并为打满带宽采用 16-lane DPP 归约与非时序访存，同时处理 NULL_BLOCK_ID 填充行与槽位隔离。
- vllm/models/kimi_k3/amd/kda.py（模块 模型层；类别 source；类型 core-logic；符号 KimiK3DeltaAttention, forward）：KimiK3DeltaAttention 的入口改造：按支持性注册解码专用 buffer 与 weight_loader，并在纯 decode 批中切换到 fused_kda_decode，是本 PR 的数据契约与派发点。
- vllm/models/kimi_k3/amd/ops/kda_decode.py（模块 内核入口；类别 infra；类型 infrastructure；符号 is_fused_kda_decode_supported, make_decode_conv1d_weight_loader, make_decode_norm_weight_loader）：集中承载运行时门控与两个专用权重加载器，决定内核何时启用、权重如何以内核友好布局加载。
- tests/models/kimi_k3/test_amd_kda_decode.py（模块 解码测试；类别 test；类型 test-coverage；符号 test_fused_kda_decode_matches_triton_chain, test_fused_kda_decode_leaves_untouched_slots_alone, test_fused_kda_decode_without_output_norm, test_fused_kda_decode_skips_null_block_padding）：数值对照测试：以 Triton 链为参照校验融合内核的输出、conv state 与递推 state，并覆盖未触及槽位与 NULL_BLOCK 填充行为，是内核正确性的主要防线。
- benchmarks/kernels/benchmark_kimi_k3_kda_decode.py（模块 微基准；类别 source；类型 benchmark；符号 _bench, _bench_graph, _bench_graph_layers, Inputs）：提供可复现的性能测量：同时支持 eager 与 CUDA-graph 计时，graph 模式按 69 层均摊启动开

销，是判断融合内核在服务器场景是否值得的决策工具。

- CMakeLists.txt（模块 构建脚本；类别 config；类型 build-config）：构建接线：仅在 HIP arch 列表含 gfx950 时编译融合内核并暴露 fused_kda_decode 符号，是多 arch 构建正确性的关键。

关键符号：kda_decode_fusion_kernel, is_fused_kda_decode_supported, make_decode_conv1d_weight_loader, make_decode_norm_weight_loader, KimiK3DeltaAttention.forward

关键源码片段

csrc/libtorch_stable/kimi_k3/fused_kda_decode_kernel_rocm.cu

新增 800 行 HIP 融合内核，是本次性能提升的核心；实现 conv1d update、gated delta-rule 递推与门控 RMSNorm 的单 launch 融合，并为打满带宽采用 16-lane DPP 归约与非时序访存，同时处理 NULL_BLOCK_ID 填充行与槽位隔离。

```
// gdn_attn.py 会把 cuda-graph decode 批的尾部用 NULL_BLOCK_ID (0) 填充。
// 被替换的 Triton 链对填充行有相同约定：递推内核把输出清零后返回，
// causal_conv1d_update 不碰 conv state。若不处理，每个填充行都会对
// slot 0 的 state 做一次无意义的读改写。slot 是 block 级统一的量，
// 因此这里提前 return 不会破坏后续 __syncthreads() 的配对。
if (slot <= 0) {
    if (tid < kDimV) {
        out[static_cast<int64_t>(i_n) * kLocalDim + i_hv * kDimV + tid] =
            bf16_store(0.0f);
    }
    return;
}
```

vllm/models/kimi_k3/amd/kda.py

KimiK3DeltaAttention 的入口改造：按支持性注册解码专用 buffer 与 weight_loader，并在纯 decode 批中切换到 fused_kda_decode，是本 PR 的数据契约与派发点。

```
        conv_state, recurrent_state = constant_caches
        # conv_state 必须是 (..., dim, width - 1) 布局：DS 布局直接满足，SD 布局要转置
        if not is_conv_state_dim_first():
            conv_state = conv_state.transpose(-1, -2)

        # 融合内核只接管"纯 decode、无 spec、无 prefill"的批：
        # 三个条件缺一不可，否则回退到下方逐 token 的 Triton 链
        if (self.decode_conv1d_weight is not None
                and self.decode_norm_weight is not None
                and spec_sequence_masks is None
                and m.num_prefills == 0
                and m.num_decodes > 0):
            assert non_spec_state_indices_tensor is not None
            ops.fused_kda_decode(
                x=mixed_qkv,
```

```
            weight=self.decode_conv1d_weight,
            bias=self.conv1d.bias,
            conv_state=conv_state,
            raw_g=g1,
            raw_beta=beta,
            A_log=self.A_log,
            dt_bias=self.dt_bias,
            state_indices=non_spec_state_indices_tensor[:num_actual_tokens],
            state=recurrent_state,
            out=core_attn_out[:, :num_actual_tokens],
            lower_bound=self.gate_lower_bound,
            output_gate=g2[:num_actual_tokens],
            norm_weight=self.decode_norm_weight,
            norm_eps=self.o_norm.eps,
        )
        return
```

vllm/models/kimi_k3/amd/ops/kda_decode.py

集中承载运行时门控与两个专用权重加载器，决定内核何时启用、权重如何以内核友好布局加载。

```
# SPDX-License-Identifier: Apache-2.0
"""ROCm 侧融合 KDA 解码内核的入口：运行时门控 + 权重加载器。

内核（csrc/libtorch_stable/kimi_k3/fused_kda_decode_kernel_rocm.cu）把纯非投机
decode 批原本的三次 Triton 启动与两次拷贝合并为一次，因此需要 width-major 的
conv 权重与 fp32 norm 权重，二者都在加载期由下面两个 loader 准备好。
"""
from collections.abc import Callable

import torch

from vllm.logger import init_logger
from vllm.model_executor.layers.mamba.mamba_utils import is_conv_state_dim_first
from vllm.model_executor.model_loader.weight_utils import default_weight_loader

logger = init_logger(__name__)

# 内核实例化的 head 数（Kimi-K3 共 96 个 KDA head，覆盖 TP1/2/4/8）。
SUPPORTED_NUM_HEADS = (12, 24, 48, 96)


def is_fused_kda_decode_supported(
    num_heads: int, head_dim: int, conv_width: int, num_spec: int,
    input_dtype: torch.dtype, conv_state_dtype: torch.dtype,
) -> bool:
    """判断本层、本设备是否可用融合解码内核。"""
    from vllm.platforms.rocm import on_gfx950 # 只有 ROCm 平台才存在该符号
```

```
    if (num_heads not in SUPPORTED_NUM_HEADS or head_dim != 128
            or conv_width != 4 or num_spec != 0
            or input_dtype != torch.bfloat16
            or conv_state_dtype != torch.bfloat16
            or is_conv_state_dim_first()
            or not hasattr(torch.ops._C, "fused_kda_decode")):
        return False
    # TODO: 其余架构尚未验证，先只放行 gfx950
    return on_gfx950()


def make_decode_conv1d_weight_loader(
    dims: list[int], tp_size: int, tp_rank: int,
    decode_conv1d_weight: torch.Tensor | None,
) -> Callable[..., None]:
    """加载 packed conv 权重，并镜像一份 width-major fp32 副本。

    内核按 [qkv, width, channel] 索引权重（channel 连续）；Triton 的 prefill
    与回退 decode 仍用 [channel, width] 布局，二者互不影响。
    """
    sharded_dims = [dim // tp_size for dim in dims]

    def weight_loader(param: torch.Tensor, loaded_weight: torch.Tensor,
                      loaded_shard_id: int) -> None:
        if loaded_weight.dim() == 2:
            loaded_weight = loaded_weight.unsqueeze(1)
        shard_size = sharded_dims[loaded_shard_id]
        source_start = tp_rank * shard_size
        target_start = sum(sharded_dims[:loaded_shard_id])
        loaded_shard = loaded_weight[source_start:source_start + shard_size]
        param.data[target_start:target_start + shard_size].copy_(loaded_shard)
        if decode_conv1d_weight is not None and not param.is_meta:
            decode_conv1d_weight[loaded_shard_id].copy_(
                loaded_shard.squeeze(1).transpose(0, 1))

    return weight_loader


def make_decode_norm_weight_loader(
    decode_norm_weight: torch.Tensor,
) -> Callable[..., None]:
    """加载门控 norm 权重，并镜像 fp32 副本供内核 epilogue 使用。"""

    def weight_loader(param: torch.Tensor, loaded_weight: torch.Tensor) -> None:
        default_weight_loader(param, loaded_weight)
        if not param.is_meta:
            decode_norm_weight.copy_(param.data)

    return weight_loader
```

评论区精华

review 中 tjtanaa 提出三点：

- 在 kimi_gdn_linear_attn.py 的 diff 上要求“please fix the prefix commit”（提交信息 / 前缀问题，未展开说明）。
- 询问既然 PR#50592（ROCm KDA 重构）已合并，是否可以把相关逻辑全部移到 vllm/models/kimi_k3/amd/kda.py；从提交历史“adapt to rocm kda refactor”与最终文件清单看已落实。
- 测试文件要求 gfx950 skip 判断必须先 current_platform.is_rocm() 再 import on_gfx950，作者回复“Guarded the tests for gfx950”，已解决。 Fangzhou-Ai 在 issue 侧补充两条关键信息：一是请作者用更新后的 recipes（recipes#733）重新基准；二是给出 8×MI355X 固定 cohort trace 的预算分析——69 层 KDA 在 C16/C24 下基线链合计约 1.80 ms / 2.09 ms，这是融合收益的理论上限（实际仍要搬移递推 state），结合 C16/C24 附近 1.8-2.3× 微基准加速比，说明 e2e ~7% 是合理预期而不是夸大。
- 提交前缀问题（please fix the prefix commit）（style）: 未在评论区明确澄清；提交历史后续以 Merge branch main 与 adapt to rocm kda refactor 等信息推进。
- PR#50592 合并后是否可将逻辑收敛到 amd/kda.py (design): 提交历史中的 adapt to rocm kda refactor 以及最终文件清单（新增 kda_decode.py、修改 kda.py）表明已按此方向落实。
- 测试文件需在 is_rocm() 之后才 import on_gfx950 (testing): 作者回复 Guarded the tests for gfx950，head 版本已按 _on_gfx950() 的先后顺序实现。
- 使用更新后的 recipes 重测（recipes#733）（question）: 评论区未见作者明确回应；PR 最终以 body 中的 MI355X 结果合并。
- 固定 cohort trace 的关键路径预算分析 (performance): 作为信息性补充，确认 PR 收益量级与预算分析一致。

风险与影响

- 风险：核心风险集中在：
  - gfx950 专属：内核只在 gfx950 上实测，is_fused_kda_decode_supported 末尾带 TODO 注释；其它 ROCm arch 由 on_gfx950() 运行时拦截，风险是回退路径与内核路径行为不一致（例如 conv state 布局判断 is_conv_state_dim_first() 在部分设备上可能为 True 导致内核永不启用）。
  - 数值等价：测试用 3e-2 / 2e-3 容差并注明融合内核在 norm 前保持 fp32，Triton 链会中间 round 到 BF16；真实权重分布下累计误差可能超过测试幅值，尤其 spec decode 场景不在覆盖内。
  - 权重加载器替换：o_norm.weight 的 weight_loader 被替换为同时 mirror fp32 副本的版本，若后续叠加量化（如 FP8）或 LoRA 等自定义 loader，可能出现冲突。
  - 测试覆盖缺口：SUPPORTED_NUM_HEADS 包含 48（TP=2），但 test_fused_kda_decode_matches_triton_chain 只测 12/24/96；num_tokens 的 7 也未覆盖 CUDA graph 常见的 padding 形状，不过 padding 行为有专门测试。

    - 构建面：含 gfx950 的多 arch 构建会对每个 arch 编译该 800 行源文件，增加编译时间；若在非 ROCm 平台启用 HIP 构建会出现问题，但 CMake 已限定 HIP 分支。
    - CUDAGraph 兼容性：作者测试命令显式关闭 VLLM_USE_BREAKABLE_CUDAGRAPH，融合路径与 breakable cudagraph 的组合未在 PR 内验证；benchmark 的 --graph 模式表明内核本身支持 graph replay，但服务器端完整路径仍需观察。
- 影响：影响范围：
    - 用户：gfx950 (MI355X 等) 上跑 Kimi-K3、且批内无非 spec / prefill 混合的 decode 用户将获得约 6.8% 的 e2e 输出吞吐提升与 6.8% 的 TPOT 改善；其它 ROCm 设备与混合批自动回退，行为不变。
    - 系统：新增 800 行 HIP 内核进入 libtorch_stable 构建，仅 gfx950 构建产物增大；新增两个非持久 buffer 与两个 loader 分支，内存占用可忽略。
    - 团队：ROCm 侧 Kimi-K3 性能专项的里程碑之一，为后续把融合思路推广到 spec decode 或其它线性注意力模型提供了内核范本与可复用的测量工具（benchmark 脚本）。CI 需要 gfx950 机器才能执行新测试，非 gfx950 CI 均为 skip。
    - 风险标记：gfx950 专属内核，数值等价依赖容差，权重加载路径变更，多 arch 构建编译面，breakable cudagraph 未验证

关联脉络

- PR #50592 （标题未在上文中提供）：review 中 tjtanaa 提到的 ROCm KDA 重构 PR，本 PR 提交历史有 adapt to rocm kda refactor，直接影响最终代码落点。
- PR #51738 [Perf] Avoid more GPU<->CPU syncs on the model execution path: 同为 V1 decode 热路径的性能优化（消除 GPU↔CPU 同步），与本 PR 的 kernel 融合方向互补。
- PR #48223 [Perf][ROCm] Dual-stream decode with hipgraphs: 同为 ROCm 解码性能专项（共享专家双流 decode + hipgraph），与本次融合内核同属 ROCm decode 性能优化线。