

PR #40977 完整报告

vllm-project/vllm

[ROCm][Kernel] Add HybridW4A16LinearKernel: Triton prefill + HIP skinny decode

合并时间: 2026-07-14 11:22

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/40977>

执行摘要

- 一句话: ROCm RDNA 混合 W4A16 内核, 解码 HIP 瘦 GEMM + 预填 Triton
- 推荐动作: 建议精读以理解混合内核设计模式 (统一权重布局、批量感知路由) 和性能分析工具使用。但需注意 review 中指出的 LDS 硬编码、group_size 64 缺失和 bias 索引问题, 应提交 follow-up PR 修复后再投入生产。

功能与动机

PR body 指出 ROCm 上 W4A16 量化模型 (如 Qwen3-4B、Gemma-3-4B) 的解码时延远高于预期, 现有 Triton kernel 对解码批量大小 1 的场景欠优。作者通过将解码专用 HIP skinny GEMM 与预填 Triton kernel 组合, 实现全量速度提升。性能表显示 TPOT 加速比可达 3-4 倍。

实现拆解

1. 内核注册与平台守卫: 在 `vllm/model_executor/kernels/linear/mixed_precision/__init__.py` 中加入 `RDNAHybridW4A16LinearKernel`, 通过 `_on_gfx12x/_on_gfx1x` 等守卫限制仅在 RDNA 平台启用。
2. 自定义算子绑定: 在 `vllm/_custom_ops.py` 中注册 `wvSplitK_int4_g` 算子 (调用 `_rocm_C.wvSplitK_int4_g`) 及其 fake 实现, 确保 `torch.compile` 兼容。
3. HIP 瘦解码内核: 新增 `csrc/rocm/skinny_gemms_int4.cu` 约 800 行 C++ 代码, 实现 `wvSplitK_int4_hf_sml_` (LDS 小规模) 与 `wvSplitK_int4_hf_` (大规模) 两个模板, 使用 DPP 缩减与 magic-number 反量化。
4. Triton 预填内核: 在 `rdna_hybrid_w4a16.py` 中用 Triton JIT 实现 `_triton_w4a16_skinny_fmt_kernel`, 读取 `[N, K//8]` int32 权重, 在寄存器级别解混洗 ExLlama 顺序。
5. 混合调度函数: `_rdna_hybrid_w4a16_apply_impl` 根据 M 值 (且检查 `KxM` 是否超过 LDS 容量) 选择 HIP 或 Triton 路径, 并添加 profiling 范围。
6. 测试与基准: `tests/kernels/quantization/test_rdna_hybrid_w4a16.py` 提供 Python 参考实现对比, 覆盖组大小、对称性、多批量。`benchmarks/kernels/benchmark_rdna_hybrid_w4a16_gemm.py` 提供 TFLOP/s 基准图。修改 `test_w4a16_kernel_selection.py` 验证内核选择优先级。

关键文件:

- `vllm/model_executor/kernels/linear/mixed_precision/rdna_hybrid_w4a16.py` (模块 混合内核; 类别 `source`; 类型 `core-logic`; 符号 `_on_gfx12x`, `_on_gfx1x`, `_on_gfx1151`, `_triton_w4a16_skinny_fmt_kernel`) : 新内核的主 Python 实现, 包含平台守卫、Triton 预填内核、HIP 调度和自定义 op 注册。
- `benchmarks/kernels/benchmark_rdna_hybrid_w4a16_gemm.py` (模块 基准测试; 类别 `source`; 类型 `core-logic`; 符号 `prepare_hybrid_weights`, `benchmark`, `run`, `prepare_shapes`) : 提供 ROCm 混合内核的独立基准, 量化性能提升并生成 TFLOP/s 图。
- `tests/kernels/quantization/test_rdna_hybrid_w4a16.py` (模块 测试覆盖; 类别 `test`; 类型 `test-coverage`; 符号 `_rdna_hybrid_w4a16_reference`, `test_rdna_hybrid_w4a16_apply_matches_reference`, `test_rdna_hybrid_w4a16_apply_with_bias`, `test_pack_int4_exllama_shuffle_layout`) : 核心正确性测试, 通过 Python 参考实现对比验证 HIP 和 Triton 两条路径。
- `vllm/_custom_ops.py` (模块 自定义操作; 类别 `source`; 类型 `core-logic`; 符号 `wvSplitK_int4_g`, `_wvSplitK_int4_g_fake`) : 注册 W4A16 自定义算子及其 fake 算子, 是 Python 侧与 C++ 内核的桥梁。
- `tests/kernels/quantization/test_w4a16_kernel_selection.py` (模块 内核选择; 类别 `test`; 类型 `test-coverage`; 符号 `_expected_rocm_kernel`, `test_choose_mp_linear_kernel_uint4b8`, `test_choose_mp_linear_kernel_uint4_asymmetric`) : 验证 ROCm 上内核选择优先级 (Hybrid > Triton), 确保新内核被正确调度。
- `csrc/rocm/skinny_gemms_int4.cu` (模块 HIP 内核; 类别 `other`; 类型 `core-logic`) : HIP 瘦解码内核的 C++ 实现, 包含 `sml` 和 `med` 两个模板, 是解码性能的关键。

关键符号: `_rdna_hybrid_w4a16_apply_impl`, `triton_w4a16_skinny_fmt_gemm`, `wvSplitK_int4_g`, `_triton_w4a16_skinny_fmt_kernel`, `pack_int4_exllama_shuffle`, `_rdna_hybrid_w4a16_apply_fake`, `benchmark`, `prepare_hybrid_weights`, `_rdna_hybrid_w4a16_reference`

关键源码片段

`vllm/model_executor/kernels/linear/mixed_precision/rdna_hybrid_w4a16.py`

新内核的主 Python 实现, 包含平台守卫、Triton 预填内核、HIP 调度和自定义 op 注册。

```
# SPDX-License-Identifier: Apache-2.0
# Copyright contributors to the vLLM project
"""
Hybrid W4A16 kernel: Triton for prefill, HIP skinny for decode.

Routes based on batch size M:
    M <= MAX_SKINNY_BATCH_SIZE: HIP skinny GEMM (wvSplitK_int4_g)
    M > MAX_SKINNY_BATCH_SIZE: Triton W4A16 fused dequant GEMM

Stores the weights ONCE as int8 [N, K//2] (ExLlama shuffle packed). Both
paths read this single buffer.
"""

# 平台检测辅助函数 (惰性导入, 防止非 ROCm 平台报错)
```

```

def _on_gfx1x() -> bool:
    if not current_platform.is_rocm():
        return False
    from vllm.platforms.rocm import on_gfx1x
    return on_gfx1x()

def _on_gfx1151() -> bool:
    if not current_platform.is_rocm():
        return False
    from vllm.platforms.rocm import on_gfx1151
    return on_gfx1151()

# 最大批量阈值: HIP 瘦内核最多支持 M=5
MAX_SKINNY_BATCH_SIZE = 5
# 每个 workgroup 的 LDS 容量 (64 KiB)
LDS_CAPACITY_ELEMENTS = 64 * 1024 // 2 # 32768 fp16 元素

# Triton 预填内核 (读取 [N, K//8] int32 权重)
@triton.jit
def _triton_w4a16_skinny_fmt_kernel(
    a_ptr, b_ptr, scales_ptr, zp_ptr, c_ptr,
    M, N, K, K8, num_groups, group_size,
    ZP_BIAS: tl.constexpr, HAS_ZP: tl.constexpr,
    BLOCK_M: tl.constexpr, BLOCK_N: tl.constexpr, BLOCK_K: tl.constexpr,
):
    # 每个 int32 打包 8 个 K 值, 使用 ExLlama 混洗
    # 在寄存器中解混洗并反量化, 与 scale 相乘后累加
    ... # 核心循环与解包逻辑
    accumulator = tl.zeros((BLOCK_M, BLOCK_N), dtype=tl.float32)
    for k_start in ...:
        # 加载激活、打包权重、缩放因子
        # 使用 shifts 从 int32 中提取 nibble 并重构
    # 写回结果

```

tests/kernels/quantization/test_rdna_hybrid_w4a16.py

核心正确性测试, 通过 Python 参考实现对比验证 HIP 和 Triton 两条路径。

```

def _rdna_hybrid_w4a16_reference(
    x_mk: torch.Tensor, # [M, K] fp16/bf16
    w_int4_nk: torch.Tensor, # [N, K] int32 raw uint4
    scales_nkg: torch.Tensor, # [N, K//G] fp16/bf16
    zp_nkg: torch.Tensor | None, # 零点 (None 表示对称)
    group_size: int,
    bias: torch.Tensor | None,
) -> torch.Tensor:
    # 反量化权重: (raw - zero_point) * scale
    # 并在 fp32 精度下执行矩阵乘法
    s_full = scales_nkg.repeat_interleave(group_size, 1).float()
    if zp_nkg is None:

```

```

        z_full = torch.full_like(s_full, 8.0) # 对称偏置
    else:
        z_full = zp_nkg.repeat_interleave(group_size, 1).float()
    w_fp = (w_int4_nk.float() - z_full) * s_full
    out = x_mk.float() @ w_fp.t()
    if bias is not None:
        out += bias.float()
    return out.to(x_mk.dtype)

@pytest.mark.parametrize("M", [1, MAX_SKINNY_BATCH_SIZE, MAX_SKINNY_BATCH_SIZE+1,
64])
def test_rdna_hybrid_w4a16_apply_matches_reference(dtype, group_size, has_zp, M):
    # 构造随机输入, 调用自定义 op ``torch.ops.vllm.rdna_hybrid_w4a16_apply``
    # 结果与参考实现对比, 允许 2e-2 容差
    ...

```

评论区精华

命名统一: tjtanaa 建议使用 `RDNAHybridW4A16LinearKernel` 避免混淆, 作者采纳并重命名文件及符号。平台安全导入: tjtanaa 指出直接 `from vllm.platforms.rocm` 会导致非 ROCm 平台报错, 作者改为惰性导入警卫函数。LDS 硬编码限制: gemini-code-assist 指出 `LDS_SIZE` 硬编码 64KB 与 host 侧动态检查不匹配, 可能导致 CDNA 架构内存损坏, 此问题未见明确修复。group_size=64 缺失: depthfirst-app 发现 `WVSPLIT_INT4G_GS` 宏仅实例化 32 和 128, 遗漏 64, 输入 64 时会静默使用错误参数, 状态未确认修复。bias 越界索引: depthfirst-app 指出偏置索引使用 `*M` 而非 `*Bx`, 可能越界, 同样未确认修复。

- 命名统一: `HybridW4A16LinearKernel` → `RDNAHybridW4A16LinearKernel` (design): 已解决, 文件及符号已更新。
- 平台安全导入: 惰性加载平台检测 (design): 已解决, 代码使用守卫函数。
- LDS 硬编码 64KB 与动态检查不匹配 (correctness): 未被显式修复, 作者未回应。最终 PR merged, 但风险依然存在。
- `WVSPLIT_INT4G_GS` 宏缺少 group_size=64 (correctness): 未被显式修复, 作者未回应。PR merged 但该问题可能影响使用 group_size=64 的模型。

bias 索引越界 (使用 `*M` 而非 `*Bx`)(correctness)未被修复, PR merged 但该风险仍需关注。

风险与影响

- 风险:
 1. C++ 内核 `LDS_SIZE` 硬编码 64 KB, 不兼容 LDS 更大的架构 (如 MI300X 128 KB), 即使仅限 RDNA 仍存在潜在风险。
 2. `WVSPLIT_INT4G_GS` 宏缺少 group_size=64 分支, 导致输入 64 时静默使用 128 的错误参数, 可能产生错误的解量化结果。
 3. bias 索引使用 `*M` 而非 `*Bx`, 可能越界读取。
 4. 仅针对 Strix Halo (gfx1151) 调优, 其他 RDNA3/4 GPU 性能不达预期, 且缺少自动化覆盖。

5. 新增约 800 行 C++ 和 560 行 Python 核心代码，增加维护负担。 - 影响：用户侧：ROCm 用户使用 W4A16 量化模型（如 Qwen3-4B、Gemma-3-4B）时，解码性能提升显著，TPOT 加速 2-4 倍，TTFT 加速 1.2-2 倍。系统侧：增加约 560 行 Python 和 800 行 C++，内核选择优先级在 RDNA 上高于 TritonW4A16LinearKernel，且自动应用于符合组大小条件的模型。团队侧：维护新的 C++ 和 Triton 内核，但设计上共享权重布局，降低了冗余。 - 风险标记：C++ 内核硬编码 LDS, group_size=64 缺失，bias 索引越界，仅限 RDNA 测试，新内核维护负担

关联脉络

- PR #42562 [Perf][Feat] Add generic cuteDSL LL BF16 router (GEMM): 同为在 ROCm 上新增高性能 GEMM 内核，展示持续的内核优化趋势。
- PR #48287 add pad-aware swiglu limit kernel: 同为新增混合精度 GEMM 内核，采用类似性能优化思路。