PR #46117 完整报告

vllm-project/vllm

[ROCm][Perf] MXFP8 dense-linear + grouped-MoE GEMM optimizations for MiniMax-M3

合并时间：2026-07-08 12:03

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/46117

执行摘要

- 一句话：AMD 上 MXFP8 GEMM 形状自适应 tile 选择与 MoE swizzle
- 推荐动作：值得精读。展示了如何通过硬件感知的 tile 选择（occupancy-driven、K-divisibility）和内存层次 swizzle 技优化 Triton 内核。_select_cfg 的设计范式可复用于其他模型和 kernel。

功能与动机

PR 原文指出："Numerically-equivalent Triton-kernel optimizations for the native MXFP8 path of MiniMax-M3 on AMD MI355X"，并强调原有固定 tile 策略无法适应 TP 分片形状，需要通过形状自适应 tile selector 和 MoE swizzle 提升 GEMM 效率。

实现拆解

1. Dense linear GEMM tile selector (rocm_native.py): 新增 _select_cfg(M, N, K) 函数，根据 token 数 M、输出维度 N、内维度 K 动态选择 BLOCK_M, BLOCK_N, BLOCK_K, num_warps, num_stages。decode 阶段（M≤64）使用窄 N 宽 K tile；中间 M 且窄 N 继续窄 tile；大 M 时根据占用率和 K 大小选择 128×128 或更优 tile。所有 BLOCK_K 确保整除 K。
2. MoE grouped GEMM swizzle (mxfp8_native_moe.py): 修改 _mxfp8_grouped_gemm_kernel，将一维 program ID 映射改为二维超分组：GROUP_SIZE_M × grid_n 超级块，使同一专家内的连续程序共享 A 行和 B 列 tile，减少 L2 重访。新增 _select_cfg 统一配置 GROUP_SIZE_M、BLOCK_K、num_warps 等。
3. Launcher 更新: _grouped_gemm_mxfp8 调用 _select_cfg 获取配置，并传递 GROUP_SIZE_M 给内核；网格尺寸基于 sorted_token_ids.shape[0]（EM）确保 swizzle 一致性。
4. 测试配套 (test_minimax_m3_amd_ops.py): 新增 _ref_grouped_gemm 纯 PyTorch 参考函数和 test_mxfp8_grouped_gemm_native 测试用例，直接验证 _grouped_gemm_mxfp8 在两种调用模式下的数值正确性（tol 5e-2）。
5. 清理: 移除争议性 preshuffle 选项和不必要的代码格式变动，增加 K 整除性守卫。

关键文件：

- vllm/model_executor/kernels/linear/mxfp8/rocm_native.py（模块 内核层；类别 source；类型 core-logic；符号 _select_cfg）：核心 dense linear GEMM 优化：用形状自适应

_select_cfg 替换固定 tile 策略，是 prefill / decode 吞吐提升的关键。

- vllm/model_executor/layers/fused_moe/experts/mxfp8_native_moe.py（模块 专家层；类别 source；类型 core-logic；符号 _select_cfg, _mxfp8_grouped_gemm_kernel）：MoE GEMM 内核优化：添加 GROUP_SIZE_M swizzle 减少 A 重读，配合 _select_cfg 统一 tile 选择，提升 grouped MoE 性能。
- tests/kernels/test_minimax_m3_amd_ops.py（模块 测试；类别 test；类型 test-coverage；符号 _ref_grouped_gemm, test_mxfp8_grouped_gemm_native）：新增对 _grouped_gemm_mxfp8 的直接测试，通过纯 PyTorch 参考确保优化后的 Triton kernel 数值正确。

关键符号：_select_cfg, _mxfp8_grouped_gemm_kernel, _ref_grouped_gemm, test_mxfp8_grouped_gemm_native

关键源码片段

vllm/model_executor/kernels/linear/mxfp8/rocm_native.py

核心 dense linear GEMM 优化：用形状自适应 _select_cfg 替换固定 tile 策略，是 prefill / decode 吞吐提升的关键。

```
# vllm/model_executor/kernels/linear/mxfp8/rocm_native.py

def _select_cfg(M, N, K):
    """
    根据 M (token 数), N (输出维度), K (内维度) 选择最佳 tile 配置。
    BLOCK_K 必须整除 K，否则 unmasked K-loop 可能越界。
    """
    # ----- 解码阶段 (M <= 64) -----
    # 小 M 是带宽 / 占用率瓶颈，通过窄 BLOCK_N (16) + 大 BLOCK_K 提升效率
    if M <= 64:
        if K % 1024 == 0: # K=2048, 6144 -> 最佳为 16x16x1024
            return 16, 16, 1024, 2, 2
        if K % 512 == 0:
            return 16, 16, 512, 2, 3
        if K % 256 == 0: # K=768 (shared_down) -> 16x32x256
            return 16, 32, 256, 4, 3
        return 16, 32, 128, 4, 3

    # ----- 中间阶段 (64 < M <= 256) 且 N 较窄 -----
    # 例如 TP=8 时 fused-qkv 本地 N=1536，继续使用窄 tile 直到 M 较大
    if (M <= 256 and N <= 1280) or (M <= 128 and N <= 1536):
        if K % 1024 == 0:
            return 16, 16, 1024, 2, 2
        if K % 512 == 0:
            return 16, 16, 512, 2, 3
        if K % 256 == 0:
            return 16, 16, 256, 2, 3
        return 16, 16, 128, 2, 3
```

```
    # ----- 大 M / 大 N 阶段 -----
    # 使用 occupancy 评估：grid (M/256)*(N/128) 决定是否采用 tall tile
    occ = triton.cdiv(M, 256) * triton.cdiv(N, 128)

    if K <= 1024: # 短 K ( 比如 shared_down 的 K=384/768)
        if M <= 256:
            if K % 256 == 0:
                return 64, 64, 256, 8, 2
            return 64, 64, 128, 8, 2
        # 大 prefill 使用 128x128x256（triton 3.6 的 256x128x256 在 3.7 上资源溢出）
        if M >= 4096 and K >= 1024 and K % 256 == 0 and occ >= 256:
            return 128, 128, 256, 8, 3
        return 128, 128, 128, 8, 3

    # K > 1024（常规 prefill）：优先 BLOCK_K=256，否则 fallback 到 128
    if K % 256 == 0:
        if M >= 4096 and occ >= 256:
            return 128, 128, 256, 8, 3
        return 64, 128, 256, 8, 3
    return 64, 128, 128, 8, 3
```

vllm/model_executor/layers/fused_moe/experts/mxfp8_native_moe.py

MoE GEMM 内核优化：添加 GROUP_SIZE_M swizzle 减少 A 重读，配合 _select_cfg 统一 tile 选择，提升 grouped MoE 性能。

```
# vllm/model_executor/layers/fused_moe/experts/mxfp8_native_moe.py

def _select_cfg(M, N, K, block_m):
    """MoE 专用的 tile 选择：BLOCK_K 优先 256（整除 K），GROUP_SIZE_M=4 实现 XCD 友好的
    swizzle。"""
    BLOCK_K = 256 if K % 256 == 0 else 128
    return {
        "BLOCK_N": 128,
        "BLOCK_K": BLOCK_K,
        "GROUP_SIZE_M": 4, # 超分组大小，将相邻 program 聚 => 常驻专家数据
        "num_warps": 8,
        "num_stages": 2,
    }

@triton.jit
def _mxfp8_grouped_gemm_kernel(...):
    pid = tl.program_id(0)
    # 从 EM (sorted_token_ids.shape[0]) 计算网格维度，而非运行时 num_post
    num_pid_m = tl.cdiv(EM, BLOCK_M)
    num_pid_n = tl.cdiv(N, BLOCK_N)
    if GROUP_SIZE_M == 1:
        pid_m = pid % num_pid_m
        pid_n = pid // num_pid_m
    else:
```

```
        # 超分组：GROUP_SIZE_M × num_pid_n 个 program 构成一个组
        num_pid_in_group = GROUP_SIZE_M * num_pid_n
        group_id = pid // num_pid_in_group
        first_pid_m = group_id * GROUP_SIZE_M
        group_size_m = min(num_pid_m - first_pid_m, GROUP_SIZE_M)
        pid_m = first_pid_m + ((pid % num_pid_in_group) % group_size_m)
        pid_n = (pid % num_pid_in_group) // group_size_m
    # 之后的加载、计算逻辑不变，但 A 行和 B 列 tile 在组内常驻缓存
```

tests/kernels/test_minimax_m3_amd_ops.py

新增对 _grouped_gemm_mxfp8 的直接测试，通过纯 PyTorch 参考确保优化后的 Triton kernel 数值正确。

```
# tests/kernels/test_minimax_m3_amd_ops.py

def _ref_grouped_gemm(a_deq, w_deq, topk_ids, a_div, num_valid, mul_weight=None):
    """纯 PyTorch 参考：对每个有效 token 做 an @ w[expert].T，可选乘以权重。"""
    eids = topk_ids.reshape(-1)
    n = w_deq.shape[1]
    out = torch.empty(num_valid, n, dtype=torch.float32, device=a_deq.device)
    for tid in range(num_valid):
        e = int(eids[tid].item())
        out[tid] = a_deq[tid // a_div].float() @ w_deq[e].float().T
        if mul_weight is not None:
            out[tid] *= float(mul_weight[tid].item())
    return out

@requires_gfx950
@pytest.mark.parametrize("T,N,K,E,top_k", [(8, 256, 128, 8, 2), (5, 512, 256, 16, 4)])
@pytest.mark.parametrize("weighted", [False, True])
@torch.inference_mode()
def test_mxfp8_grouped_gemm_native(T, N, K, E, top_k, weighted):
    """验证 _grouped_gemm_mxfp8 与纯参考的误差 < 5e-2
    weighted=False 对应 g1（a_div=top_k），weighted=True 对应 g2（per-token 权重）
    """
    from vllm.model_executor.layers.fused_moe.experts.mxfp8_native_moe import _grouped_
    gemm_mxfp8
    from vllm.model_executor.layers.fused_moe.moe_align_block_size import moe_align_block_
    size

    torch.manual_seed(0)
    block_m = 64
    a_div = 1 if weighted else top_k
    m_routed = T * top_k
    a_rows = m_routed if weighted else T
    a_bf16 = torch.randn(a_rows, K, device=DEVICE, dtype=torch.bfloat16) * 0.5
    w_bf16 = torch.randn(E, N, K, device=DEVICE, dtype=torch.bfloat16) * 0.1
    # 量化为 MXFP8，供 kernel 使用
    a_fp8, a_scale = _mxfp8_e4m3_quantize_torch(a_bf16, is_sf_swizzled_layout=False)
```

```
    w_fp8, w_scale = _mxfp8_e4m3_quantize_torch(w_bf16, is_sf_swizzled_layout=False)

    logits = torch.randn(T, E, device=DEVICE, dtype=torch.float32)
    topk_weights, topk_ids = logits.softmax(dim=-1).topk(top_k, dim=-1)
    mul = topk_weights.reshape(-1) if weighted else None

    sorted_ids, expert_ids, num_post = moe_align_block_size(topk_ids, block_m, E, None)
    got = _grouped_gemm_mxfp8(
        a_fp8, a_scale, w_fp8, w_scale,
        sorted_ids, expert_ids, num_post,
        m_routed, top_k, block_m, torch.bfloat16,
        a_div=a_div, mul_weight_by=mul)
    # 参考：反量化后纯 torch matmul
    a_deq = dequant_mxfp8_to_bf16(a_fp8, a_scale)
    w_deq = dequant_mxfp8_to_bf16(w_fp8, w_scale)
    ref = _ref_grouped_gemm(a_deq, w_deq, topk_ids, a_div, m_routed, mul)
    assert got.shape == (m_routed, N)
    assert _relerr(got, ref) < 5e-2
```

评论区精华

设计取舍：Hongxiayang 指出预设的 preshuffle 路径是死代码不应保留，作者移除该选项。_select_cfg 中的 K 整除性检查被 depthfirst-app[bot] 发现遗漏，可能引发 OOB 读取，作者在后续版本补充守卫。测试增强：fxmarty-amd 建议为 _grouped_gemm_mxfp8 添加非 Triton 参考测试，作者实现了 _ref_grouped_gemm 和 parametrized test。风格问题：Hongxiayang 指出 tl.store 拆成两步、提取 k_iters 变量等无实际意义，作者回退这些变更。

- Preshuffle 选项应该移除 (design): 移除了 preshuffle 相关代码。
- K 整除性检查缺失（OOB 风险） (correctness): 在相关分支增加了 K%256==0 守卫，否则 fallback 到 BLOCK_K=128。
- 增加非 Triton 参考测试 (testing): 测试已添加，覆盖两种调用模式，数值验证通过。
- 不必要的代码格式变更 (style): 代码恢复原始风格，减少 diff 噪声。

风险与影响

- 风险：BLOCK_K 整除性依赖手动检查当前覆盖的 K 值（384/768/1024/2048/6144），若未来新增 K 值未及时更新可能 OOB。精度容忍度 6e-2，GSM8K 精度不变。仅影响 AMD gfx950，其他平台无风险。
- 影响：用户可见性：仅 AMD CDNA4 (MI355X) 且使用 MiniMax-M3 模型时自动生效，无需配置变更。吞吐量提升显著（decode 最多 46%，prefill 稳定 20–30%）。团队影响：ROCm 团队受益，代码复杂度无明显增加。
- 风险标记：K 整除性依赖手动维护，仅 AMD gfx950 收益

关联脉络

- PR #47631 [Perf] Minimax M3 - Support cross-layer allreduce-norm fusion: 同为 MiniMax-M3 在 ROCm 上的性能优化，涉及不同的 kernel 融合。
- PR #47445 [BugFix] Fix ModelOpt quantization inference for fused siblings: 同样针对 MiniMax-M3 的量化修复，有文件交集（minimax_m3 模型文件）。