

PR #50949 完整报告

vllm-project/vllm

[CPU] Optimize routed FP8/MXFP4 MoE GEMM dispatch

合并时间: 2026-08-08 12:56

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/50949>

执行摘要

本 PR 重构了 CPU 上 FP8/MXFP4 量化 MoE 的 GEMM 分派: 将原先“整层只按全局平均 token 数决定是否使用 BRGEMM”的固定 crossover, 改为在每个 expert block 内按实际 M、N 逐块选择 BRGEMM (Intel AMX) 或固定行 TinyGEMM, 使集中路由能吃到 BRGEMM 的性能红利。基于 Sonnet 256/256 的 27 个基准点, 生成端吞吐几何平均提升约 8.7%, 最佳点 GPT-OSS 并发 32 达 1.5954x; 代价是 GPT-OSS 并发 64 出现 0.9867x 的轻微回退。该改动只影响 CPU 后端, 并新增了小 expert 块边界测试, 最终分派 API 与 SGL 侧保持一致以便后续 code bump-up。

功能与动机

PR body 明确指出原实现 `int64_t avg_M = std::max(int64_t(1), M * topk / E); const bool use_brgemm = can_use_brgemm<packed_t>(avg_M);` 用全局平均 token 数做整层决策——只要平均每 expert 的 token 数不超过阈值, 整层都退回 TinyGEMM; 而当平均 M 较大时, 小 expert 块也被强制走 BRGEMM, 二者都不是最优。本 PR 的目标是“Remove the fixed global crossover from routed CPU FP8/MXFP4 MoE dispatch so that concentrated moe routing can benefit from brgemm”, 并强调与 #49942 (通过不同 API 扩展 CPU FP8 W8A8 支持) 相互独立。Dense 与 shared-expert 分派保持不变, 控制回归面。

实现拆解

1. 新增按块判定的 BRGEMM 谓词: `csrc/cpu/sgl-kernels/gemm.h` 在既有单参 `can_use_brgemm<T>(int M)` 基础上新增双参重载 `can_use_brgemm<T>(int M, int N)`, 语义为“支持 BRGEMM 且 (M 超过小 GEMM 阈值, 或 N 为不完整的 tail 块)”。这是逐块分派的最小公共谓词, 后续所有决策都复用该函数。
2. 逐块分派下沉到 fused kernel: `csrc/cpu/sgl-kernels/moe_fp8.cpp` 的 `fused_experts_fp_kernel_impl` 中删除全局 `use_brgemm = can_use_brgemm(avg_M);` stage 1 (hidden @ w1) 与 stage 2 (activation @ w2) 的 tile 循环内改用 `const bool brg = can_use_brgemm<packed_t>(m_size, n_size);` 其中 `m_size` 是该 expert 块的实际 token 数、`n_size` 是实际 N 分块宽度, `tinygemm_kernel` 的 `brg` 参数随之逐块传入, 小块自然落到固定行 TinyGEMM 微内核。
3. BRGEMM 资源释放配对: 同一线程多个 tile 可能分别走 TinyGEMM 与 BRGEMM, 新增局部聚合标志 `use_brgemm != brg`, 在循环结束后统一调用 `at::native::cpublas::brgemm_release()`, 保证 AMX 状态释放与使用严格配对; 同时为 `m_size == 0` 的空块增加提前 return, 避免无意义的 A 行拷贝与 GEMM 调用。

4. 测试配套: tests/kernels/moe/test_cpu_quant_fused_moe.py 新增 `_deterministic_expert_routes(block_sizes)` 辅助函数, 用连续 token 精确构造每个 expert 的块大小; 新增 `test_w8a16_block_fp8_cpu_fused_moe_small_expert_blocks` 与 `test_mxfp4_cpu_fused_moe_small_expert_blocks`, `block_sizes` 取 (4, 5, 33), 覆盖阈值 4 两侧 (4 走 TinyGEMM、5 与 33 走 BRGEMM) 以及跨多个 BLOCK_M 的大块, 与浮点参考实现 `ref_*` 做 $1e-2$ 容差对比。
5. 演进与收敛: 提交历史显示实现经历了“枚举后端 + 集中谓词 → 按 review 简化为 bool + 双参重载”的来回: 最初引入 `PackedGemmBackend` 枚举与 `moe_uses_brgemm`, 维护者 bigPYY1151 以“减少与 SGL 上游分歧、便于 code bump-up”为由要求改回 bool, 最终 commit 恢复了 `use_brgemm` 命名并删除多余 adapter。

csrc/cpu/sgl-kernels/moe_fp8.cpp

核心分派逻辑所在: 将全局平均 M 的 BRGEMM 决策改为逐 expert block 按实际 `m_size`、`n_size` 决策, 并新增空块提前 return 与 `use_brgemm` 聚合释放标志。

```
// stage 1: intermediate_cache0 = hidden_states @ w1 (FP8/MXFP4 共用)
// 本 PR 核心改动: BRGEMM 决策从「全局平均 M」下沉到「每个 expert block 的
// 实际 m_size、n_size」, 集中路由的大块走 Intel AMX, 小块回退 TinyGEMM。
parallel_2d(MB, NB, [&](int64_t mb0, int64_t mb1, int64_t nb0, int64_t nb1) {
    int tid = get_thread_num();
    scalar_t* __restrict__ A = A_tmp + tid * BLOCK_M * K;
    // 聚合标志: 任一 tile 使用 BRGEMM 后统一 release, 保证 AMX 状态配对
    bool use_brgemm = false;

    loop_2d<packed_t>(mb0, mb1, nb0, nb1, BLOCK_N * K,
        [&](int64_t mb, int64_t nb, int64_t nb_offset) {
            int64_t n_size = std::min(2 * N - nb * BLOCK_N, BLOCK_N);

            // B 为 [K, n_size] 的 VNNI 打包格式, 按 expert 定位权重与量化 scale
            int32_t expert_id = expert_ids[mb];
            const packed_t* __restrict__ B =
                packed_w1 + expert_id * stride_e + nb * BLOCK_N * stride_n;
            const param_t* __restrict__ Bs =
                w1s + expert_id * scale_size_N * scale_size_K +
                scale_offset_per_block(nb) * scale_size_K;
            const float* __restrict__ B_bias =
                with_bias ? w1_bias + expert_id * 2 * N + nb * BLOCK_N : nullptr;

            // 每个新 expert 的首块需要解包 VNNI 权重
            int32_t pre_expert_id = mb == 0 ? -1 : expert_ids[mb - 1];
            bool do_unpack = (mb == mb0) || (expert_id != pre_expert_id);

            // 空 expert 块直接跳过, 避免无意义的 A 行拷贝与 GEMM 调用
            int64_t m_size = offsets[mb + 1] - offsets[mb];
            if (m_size == 0) {
                return;
            }
        })
```

```

// 逐块判定: M 超过小 GEMM 阈值 4 或 N 为 tail 块时走 BRGEMM
const bool brg = can_use_brgemm<packed_t>(m_size, n_size);
use_brgemm |= brg;

if (nb_offset == 0) {
    // 仅首个 N 块加载 A, 按排序后的 id 取原始 token 行
    const int32_t* A_ids = sorted_ids + mb * BLOCK_M;
    for (int64_t m = 0; m < m_size; ++m) {
        int32_t index = A_ids[m] / topk;
        copy_stub(A + m * K, input + index * K, K);
    }
}

// 同一内核同时承载 TinyGEMM 与 BRGEMM 两条路径, 由 brg 逐块切换
tinygemm_kernel<scalar_t>(
    /* A */ A,
    /* B */ B,
    /* C */ ic0 + offsets[mb] * 2 * N + nb * BLOCK_N,
    /* Btmp */ B_tmp + tid * B_tmp_size_per_thread + nb_offset * BLOCK_N * K,
    /* Ctmp */ C_tmp + tid * 2 * BLOCK_M * BLOCK_N,
    /* Bbias */ B_bias,
    /* scale */ Bs,
    /* M */ m_size,
    /* N */ n_size,
    /* K */ K,
    /* lda */ K,
    /* ldb */ n_size,
    /* ldc */ 2 * N,
    /* brg */ brg,
    /* block_size_K */ block_size_K,
    /* do_unpack */ do_unpack);
});

if (use_brgemm) {
    at::native::cpublas::brgemm_release();
}
});

```

csrc/cpu/sgl-kernels/gemm.h

新增双参 `can_use_brgemm(int M, int N)` 重载, 是逐块分派决策的公共谓词, 并被 `moe_fp8.cpp` 调用。

```

// define threshold using brgemm (intel AMX)
// 单参版本: 按行数 M 判定, 用于 dense 等非 routed 场景;
// 各量化类型共用「支持 BRGEMM 且 M > 4」的阈值。
template <>
inline bool can_use_brgemm<at::Float8_e4m3fn>(int M) {
    return brgemm_supported() && M > 4;
}

```

```
// 双参版本（本 PR 新增）：routed MoE 按每个 expert block 的实际 M、N 决策。  
// N 不等于完整 block 的 tail 块也走 BRGEMM，避免 TinyGEMM 的固定行约束；  
// 单参判断的 M > 4 阈值依然生效，小块保留 TinyGEMM 微内核。  
template <typename T>  
inline bool can_use_brgemm(int M, int N) {  
    return brgemm_supported() && (can_use_brgemm<T>(M) || N != block_size_n());  
}
```

评论区精华

维护者 bigPYJ1151 在复现收益（“Thanks @tianmu-li I can reproduce the improvement”）后提出三点设计意见，均被作者采纳：

“I think use bool is enough. I want to reduce divergence with SGL side to facilitate code bump-up.” —— 关于 `PackedGemmBackend` 枚举，最终回退为 bool 标志，降低与 SGL 上游同步成本。

“I suggest to add new `can_use_brgemm` overloads to keep consistent manner.” —— 关于替代 `moe_uses_brgemm` 谓词，最终以双参重载落地。

“I think these conditions can be checked in `fused_experts_fp_kernel_impl`, then the shortcuts in `brgemm` are not required.” —— 分派条件前移到调用方，避免内核内部冗余短路分支。

风险与影响

- 性能回退点：基准 27 个点中最差为 GPT-OSS 并发 64 的 0.9867x，整体 geomean 虽为正，但高并发下存在约 1% 抖动；合并后若有更多回退报告，阈值 4 与 BRGEMM 选择条件需重新评估。
- 阈值硬编码：M > 4 阈值直接写在 `gemm.h` 的模板特化中，FP8 与 MXFP4 共用，未来调整需同步 SGL 侧。
- BRGEMM 资源配对：`use_brgemm` 聚合标志与 `brgemm_release()` 的配对依赖循环内 brg 计算路径的完备性，空块提前 return 若漏置标志会破坏配对（当前 stage 1/stage 2 均已覆盖）。
- 测试覆盖局限：新增测试 `topk=1` 且 `K=N=128` 对称形状，未覆盖 `topk>1` 或 N 不等于 K 的组合；且测试仅在 CPU-only 条件下运行，CI 需确保 CPU 构建覆盖这些用例。
- 影响范围：所有 CPU 部署的 FP8/MXFP4 MoE 模型（如 Qwen3-Coder、GPT-OSS）都会经过新分派路径，GPU、dense、shared-expert 分派不受影响。

关联脉络

- 本 PR 在 body 中明确与 #49942（CPU FP8 W8A8 支持，不同 API 路线）解耦，二者共同构成 CPU 量化 MoE 的两个优化方向。
- 系列 PR 49347（在线 MXFP4 量化）提供了 MXFP4 MoE 路径，本 PR 则在不改 API 的前提下提升其分派效率。
- 该逻辑位于 `sgl-kernels` 目录，与 SGL 上游共享 kernel 代码；review 阶段刻意压低分派 API 的分歧，为后续 `sgl-kernels` 版本对齐（code bump-up）铺路。

- 同仓库近期 48735 (`--linear-backend` 过滤) 与 51411 (Triton MoE INT8 修复) 展现了 MoE quant kernel 选择与健壮性治理的持续投入, 本 PR 属于同一治理思路在 CPU 后端的落地。