PR #46353 完整报告

vllm-project/vllm

[CPU][Perf] Accelerate unquantized MoE for AArch64

合并时间：2026-06-24 22:14

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/46353

执行摘要

本 PR 为 AArch64 CPU 上的未量化 MoE 推理提供显著加速，通过新增基于 BFMMLA 指令的 NEON 微 GEMM kernel，并集成到 fused MoE 调度中。在 96 核 Neoverse-V2 上，gpt-oss 和 gemma4 吞吐量分别提升 2.1x 和 2.34x。同时，抽象 x86 特定的 sleef 依赖，确保跨架构兼容性。变更涉及 15 个文件，但核心逻辑集中在新增的 NEON 微 kernel 和调度适配，对非 AArch64 平台无影响。

功能与动机

AArch64 平台（如 ARM Neoverse）在未量化 MoE 场景下缺乏硬件加速，导致 gpt-oss 和 gemma4 等模型的推理性能受限。PR body 提供的基准显示：原有实现吞吐量不足，本 PR 旨在利用 AdvSIMD BFMMLA 指令实现融合 MoE kernel，以提升性能。

实现拆解

1. 新增 NEON 微 GEMM kernel(csrc/cpu/micro_gemm/cpu_micro_gemm_neon.hpp)：实现 8x8 BFMMLA 微 GEMM，支持输入矩阵 packing，并提供初始化 / 存储累加器的辅助函数（init_acc_rowpair、store_acc_rowpair）。
2. 集成到 fused MoE 调度(csrc/cpu/cpu_fused_moe.cpp)：添加 NEON_DISPATCH 宏，扩展 CPU_ISA_DISPATCH_IMPL 以包含 NEON 分支；修改 fused_moe_impl 根据 pack_a 标志调整 tile 大小，避免为 packed 输入预留过多缓存。
3. 激活函数适配：修改 swigluoai_and_mul 等激活函数，为 AArch64 提供专用的偶奇分离加载（load_even_odd）以避免 gather 操作；用通用 tanh() 替代 x86 特定的 Sleef_tanhf16_u10，确保跨架构正确性。
4. Python 层 ISA 发现(vllm/model_executor/layers/fused_moe/cpu_fused_moe.py)：在 check_grouped_gemm 中检测 ARM 架构并返回 "neon" ISA，使得 grouped GEMM 路径能为 AArch64 选择 NEON 微 kernel。
5. 基准与测试更新：更新 benchmarks/kernels/cpu/benchmark_cpu_fused_moe.py 和 tests/kernels/moe/test_cpu_fused_moe.py，支持 NEON ISA 选择；在 csrc/cpu/utils.hpp 中添加 NEON ISA 枚举，在 csrc/cpu/cpu_types_arm.hpp 中新增 load_even_odd 和取负操作。

csrc/cpu/micro_gemm/cpu_micro_gemm_neon.hpp

新增 NEON 微 GEMM kernel，实现基于 BFMMLA 指令的 8x8 微 GEMM，包含 pack 和 tile 支持，是本次加速的核心。

```
/*
 * 以下两个函数是 NEON 微 GEMM kernel 的核心辅助函数。
 * init_acc_rowpair 从 C 矩阵加载 1-2 行，并将它们交错压缩到 4 个 float32x4_t
 * 寄存器对 (acc01, acc23, acc45, acc67) 中，每个寄存器承载两行的部分元素。
 * 这种布局便于后续 BFMMLA 指令高效计算。
 * store_acc_rowpair 执行逆操作，将累加器拆分并写回 C 矩阵。
 */

// 加载 C 矩阵并初始化累加器对
FORCE_INLINE void init_acc_rowpair(float32x4_t& acc01, float32x4_t& acc23,
                                   float32x4_t& acc45, float32x4_t& acc67,
                                   const float* __restrict__ c_ptr,
                                   const int64_t ldc, const int32_t m_rows,
                                   const bool accum_c) {
  if (!accum_c || m_rows == 0) {
    // 清空为 0
    acc01 = vdupq_n_f32(0.0f);
    acc23 = vdupq_n_f32(0.0f);
    acc45 = vdupq_n_f32(0.0f);
    acc67 = vdupq_n_f32(0.0f);
    return;
  }

  // 加载行 0 和行 1 ( 或填充 0)
  const float32x4_t row0_0123 = vld1q_f32(c_ptr);
  const float32x4_t row0_4567 = vld1q_f32(c_ptr + 4);
  const float32x4_t row1_0123 =
      (m_rows == 2) ? vld1q_f32(c_ptr + ldc) : vdupq_n_f32(0.0f);
  const float32x4_t row1_4567 =
      (m_rows == 2) ? vld1q_f32(c_ptr + ldc + 4) : vdupq_n_f32(0.0f);

  // 交错：acc01 = [row0_0, row0_1, row1_0, row1_1]
  acc01 = zip1_f32x4(row0_0123, row1_0123);
  acc23 = zip2_f32x4(row0_0123, row1_0123);
  acc45 = zip1_f32x4(row0_4567, row1_4567);
  acc67 = zip2_f32x4(row0_4567, row1_4567);
}

// 将累加器对写回 C 矩阵
FORCE_INLINE void store_acc_rowpair(const float32x4_t acc01,
                                    const float32x4_t acc23,
                                    const float32x4_t acc45,
                                    const float32x4_t acc67,
                                    float* __restrict__ c_ptr,
                                    const int64_t ldc, const int32_t m_rows) {
  if (m_rows == 0) return;

  // 反交错：将 acc01/acc23 解压为行 0 的 0-3 和 4-7 元素
  vst1q_f32(c_ptr, zip1_f32x4(acc01, acc23));
```

```
    vst1q_f32(c_ptr + 4, zip1_f32x4(acc45, acc67));

    if (m_rows == 2) {
      // 写回行 1
      vst1q_f32(c_ptr + ldc, zip2_f32x4(acc01, acc23));
      vst1q_f32(c_ptr + ldc + 4, zip2_f32x4(acc45, acc67));
    }
  }
```

评论区精华

- bigPYJ1151：指出 w2_input_tile_size 在输入被 packed 后可能不再需要，因为 GEMM kernel 只看到 packed buffer。
- fadara01：同意并引入了条件分支 w2_input_buffer_size，但保留了原始计算作为未使用变量，可能是为未来解 packed 路径预留。

风险与影响

- 风险：NEON 分支通过 ARM_BF16_SUPPORT 和 __aarch64__ 条件编译隔离，不影响其他架构。pack 逻辑可能引入内存越界，但 round_up 和联动检查控制风险。sleef.h 移除后经 DiffusionGemma 测试验证。
- 影响：对 AArch64 用户性能提升 2x+，代码维护成本增加但设计清晰。无向后兼容性问题。

关联脉络

本 PR 是 vLLM CPU back-end 中对 AArch64 的首个专用 MoE 加速，与之前的 CPU fused MoE 基础框架（如 grouped GEMM）一脉相承。后续可能扩展到其他 ISA（如 SVE）或量化 MoE 的 AArch64 支持。