PR #44324 完整报告

vllm-project/vllm

[CPU][RISC-V] Add RVV micro GEMM for WNA16

合并时间：2026-06-22 20:53

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/44324

执行摘要

- 一句话：为 CPU WNA16 添加 RVV 微 GEMM 内核，加速 2.4-3.2x
- 推荐动作：值得精读。设计上清晰地分离了 ISA 枚举、微 GEMM 内核、C++ 分发、Python 前端，层次分明。RVV 内核实现中 Mx8 tile 和 K 展开的策略可推广到其他量化场景。代码注释丰富，适合作为为特定 ISA 添加微内核的参考。

功能与动机

RISC-V CPU 上现有 VEC 路径使用通用向量抽象，寄存器压力大，性能不理想。本 PR 旨在通过 RVV 专用微 GEMM 提升 WNA16 量化模型的推理速度，满足国产 RISC-V 硬件的部署需求。PR 描述中给出加速比和数值匹配结果。

实现拆解

实现分为四步：

1. ISA 枚举扩展：在 csrc/cpu/utils.hpp 中新增 ISA::RVV 枚举值，并在 get_isa() 中添加对 "rvv" 字符串的解析，为后端分发做准备。
2. RVV 微 GEMM 内核：新增 csrc/cpu/micro_gemm/cpu_micro_gemm_rvv.hpp，实现 MicroGemm<ISA::RVV, scalar_t>。核心内核函数 gemm_micro_rvv_fma_mx8_ku4 使用 Mx8 内部 tile，保持外部 N=32 的 packed weight 布局兼容；通过标量 - 向量 FMA 利用激活广播模式；K 循环按 4 展开以减少指令开销。同时提供 load_row8_b_as_f32 模板特化，支持 float/Half/BFloat16 到 float 的高效加载，并利用可选的 zvfh/zvfbfmin 扩展。
3. C++ 连接：在 csrc/cpu/cpu_wna16.cpp 中通过 #if defined(__riscv_v) 条件包含新头文件，在 cpu_gemm_wna16 的 ISA 解析分支中添加 "rvv" 映射，并实例化 MicroGemm<ISA::RVV, scalar_t> 及对应的 Dequantizer4b，复用相同的反量化逻辑。
4. Python 前端：在 vllm/model_executor/kernels/linear/mixed_precision/cpu.py 中导入 CpuArchEnum，在 _get_isa_hint() 中检测当前 CPU 架构是否为 RISC-V，若是则返回 "rvv"，确保 ops.cpu_gemm_wna16 接收到正确的 ISA 提示。

关键文件：

- csrc/cpu/micro_gemm/cpu_micro_gemm_rvv.hpp（模块 CPU 内核；类别 source；类型 core-logic；符号 TileGemmRVV, MicroGemm）：核心新增文件，实现 RVV 微 GEMM 内核，包含模板化的 load 和 FMA 内核。

- csrc/cpu/cpu_wna16.cpp（模块 CPU 推理；类别 source；类型 dependency-wiring）：桥接文件，通过条件编译包含 RVV 头文件，并在 dispatch 函数中添加 RVV 分支。
- csrc/cpu/utils.hpp（模块 CPU 工具；类别 source；类型 core-logic；符号 class）：定义 ISA::RVV 枚举并扩展 get_isa()，是分发的基础。
- vllm/model_executor/kernels/linear/mixed_precision/cpu.py（模块 量化驱动；类别 source；类型 data-contract）：Python 前端检测 RISC-V 架构并传递 "rvv" isa_hint 至 C++ 后端。

关键符号：gemm_micro_rvv_fma_mx8_ku4, load_row8_b_as_f32, cpu_gemm_wna16, _get_isa_hint, get_isa

关键源码片段

csrc/cpu/micro_gemm/cpu_micro_gemm_rvv.hpp

核心新增文件，实现 RVV 微 GEMM 内核，包含模板化的 load 和 FMA 内核。

```
// RVV 微 GEMM 内核：固定 N=8 内部 tile，K 循环展开 4 次，使用标量 - 向量 FMA。
// 该函数被 MicroGemm<ISA::RVV, scalar_t> 调用，完成 Mx8 子矩阵乘法。
template <int32_t M, typename scalar_t>
FORCE_INLINE void gemm_micro_rvv_fma_mx8_ku4(
    const scalar_t* __restrict__ a_ptr, // [M, K] 激活矩阵
    const scalar_t* __restrict__ b_ptr, // [K, 8] 权重矩阵（部分 packed）
    float* __restrict__ c_ptr, // [M, 8] 累加结果
    const int64_t lda, const int64_t ldc, // leading dimensions
    const int32_t k, const bool accum_c) {
  static_assert(0 < M && M <= 8);

  // 声明 8 行指针和累加器（使用宏展开，避免重复代码）
  #define RVV_ROWS_APPLY(OP) OP(0) OP(1) OP(2) OP(3) OP(4) OP(5) OP(6) OP(7)
  #define RVV_IF_M(i) if constexpr (M > (i))

  #define RVV_DECL_A(i) const scalar_t* __restrict__ a##i = a_ptr + (i) * lda;
  RVV_ROWS_APPLY(RVV_DECL_A)
  #undef RVV_DECL_A

  #define RVV_DECL_ACC(i) fixed_fp32x8_t acc##i;
  RVV_ROWS_APPLY(RVV_DECL_ACC)
  #undef RVV_DECL_ACC

  // 初始化累加器：若 accum_c 则加载已有值，否则清零
  #define RVV_INIT_ACC(i) \
    RVV_IF_M(i) { \
      if (accum_c) { \
        acc##i = RVVI(__riscv_vle32_v_f32, LMUL_256)(c_ptr + (i) * ldc, RVV_MGEMM_N8); \
      } else { \
        acc##i = RVVI(__riscv_vfmv_v_f_f32, LMUL_256)(0.0f, RVV_MGEMM_N8); \
      } \
    }
```

```
  RVV_ROWS_APPLY(RVV_INIT_ACC)
  #undef RVV_INIT_ACC

  // 主循环：每次处理 4 个 K 元素，减少循环开销
  int32_t k_idx = 0;
  for (; k_idx + 3 < k; k_idx += 4) {
    // 加载 b 矩阵的一行（N=8），并转换为 float
    // 实际由 load_row8_b_as_f32 根据 scalar_t 类型特化实现
    #define RVV_STEP_K(K_OFFSET) \
    { \
      fixed_fp32x8_t b = load_row8_b_as_f32<scalar_t>( \
          b_ptr + (k_idx + (K_OFFSET)) * RVV_MGEMM_B_GROUP_STRIDE); \
      RVV_FMA_ROW(0, K_OFFSET) \
      RVV_FMA_ROW(1, K_OFFSET) \
      RVV_FMA_ROW(2, K_OFFSET) \
      RVV_FMA_ROW(3, K_OFFSET) \
      RVV_FMA_ROW(4, K_OFFSET) \
      RVV_FMA_ROW(5, K_OFFSET) \
      RVV_FMA_ROW(6, K_OFFSET) \
      RVV_FMA_ROW(7, K_OFFSET) \
    }
    // RVV_FMA_ROW 使用 vfmacc 执行标量 - 向量融合乘加
    #define RVV_FMA_ROW(i, K_OFFSET) \
      RVV_IF_M(i) { \
        acc##i = RVVI(__riscv_vfmacc_vf_f32, LMUL_256)( \
            acc##i, static_cast<float>(*(a##i + k_idx + (K_OFFSET))), b, RVV_MGEMM_N8); \
      }

    RVV_STEP_K(0)
    RVV_STEP_K(1)
    RVV_STEP_K(2)
    RVV_STEP_K(3)
    #undef RVV_STEP_K
    #undef RVV_FMA_ROW
  }
  // ... 处理剩余 K 元素（省略）
}
```

csrc/cpu/cpu_wna16.cpp

桥接文件，通过条件编译包含 RVV 头文件，并在 dispatch 函数中添加 RVV 分支。

```
// cpu_wna16.cpp 中新增的 RVV 分发分支（位于 cpu_gemm_wna16 函数内）
  } else if (isa == ISA::RVV) {
    // 实例化 MicroGemm<ISA::RVV, scalar_t>
    using gemm_t = cpu_micro_gemm::MicroGemm<ISA::RVV, scalar_t>;
    if (has_zp) {
      using dequantizer_t = Dequantizer4b<scalar_t, ISA::RVV, true, false>;
      cpu_gemm_wna16_impl<scalar_t, dequantizer_t, gemm_t>(
          input.data_ptr<scalar_t>(), q_weight.data_ptr<int32_t>(),
```

```
            output.data_ptr<scalar_t>(), scales.data_ptr<scalar_t>(), zeros_ptr,
            g_idx_ptr, bias_ptr, a_m_size, b_n_size, a_k_size, a_m_stride,
            output_m_stride, scales_group_stride, zeros_group_stride, group_num,
            group_size, pack_factor);
        return;
      }
      if (use_desc_act) {
        // 类似，使用 Dequantizer4b<scalar_t, ISA::RVV, false, true>
        // ...
      } else {
        // 一般情况：无 zero-point，无 desc_act
        using dequantizer_t = Dequantizer4b<scalar_t, ISA::RVV, false, false>;
        cpu_gemm_wna16_impl<scalar_t, dequantizer_t, gemm_t>( ... );
        return;
      }
    }
```

vllm/model_executor/kernels/linear/mixed_precision/cpu.py

Python 前端检测 RISC-V 架构并传递 "rvv" isa_hint 至 C++ 后端。

```
  # cpu.py 中 _get_isa_hint 函数修改后
  def _get_isa_hint(dtype: torch.dtype) -> str:
      supports_amx = torch.cpu._is_amx_tile_supported()
      if supports_amx and dtype in (torch.bfloat16,):
          return "amx"
      elif current_platform.get_cpu_architecture() == CpuArchEnum.RISCV:
          # RISC-V 平台使用 RVV 微 GEMM 后端
          return "rvv"
      else:
          return "vec"
```

评论区精华

本 PR 无 Review 讨论，maintainer @bigPYJ1151 直接批准合并。

- 暂无高价值评论线程

风险与影响

- 风险：回归风险低：RVV 内核仅在 __riscv_v 编译时生效，不影响 x86/ARM 等其他架构。数值匹配验证（max|rvv-vec| = 0）覆盖常见 shape，反量化路径复用现有代码。 性能风险：当前 tile shape 针对 VLEN=128 调优，更长 VLEN 的 CPU 可能未达最佳，但 PR 在 follow-up 中已提及未来可调。 测试覆盖：PR 缺少与 CI 集成的自动化测试，未来重构可能有退化风险。建议在 RISC-V CI 节点上添加 tests/quantization/test_cpu_wna16.py 的回归测试。
- 影响：对用户：RISC-V CPU 上运行 W4A16 GPTQ/AWQ 模型的用户将获得 2.4-3.2x GEMM 加速，推理延迟明显降低。对其他用户无影响。对系统：新增约 230 行 C++ 内核代码，无外部依赖。对团队：需维护 RVV 内核，但 kernel 为模板化实现，与现有架构一致。

- 风险标记：仅 RISC-V 路径变更，缺少 CI 回归测试，tile shape 针对 VLEN=128 优化

关联脉络

- PR #46313 [Bugfix] Reject matryoshka embedding dimensions above hidden size: 同为 CPU 平台相关修改，但无直接功能关联。
- PR #46216 [CPUOffloadingManager] Maintain evictable list in LRUCachePolicy: 同为 CPU 后端优化，但领域不同（offload vs GEMM）。