PR #40119 完整报告

vllm-project/vllm

[CPU][RISC-V] Add RVV-optimized attention kernels for RISC-V Vector Extension

合并时间：2026-05-15 12:08

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/40119

执行摘要

- 一句话：为 CPU 后端添加 RISC-V RVV 优化注意力核 (2.3x-3.7x)
- 推荐动作：该 PR 是跨平台支持的重要里程碑，值得架构师和 CPU 后端开发者精读。核心设计决策包括：使用编译时预处理器门控解决 ISA 可变性问题、通过 /proc/cpuinfo 运行时检测与 CMake 编译保持一致、以及利用空标签结构体绕开 GCC 模板编译错误的优雅方法。建议关注后续 VLEN 通用化重构。

功能与动机

RISC-V 平台缺乏高效的 LLM 推理支持。PR body 明确指出目标是“enable efficient LLM inference on RISC-V platforms”，并给出了 SG2044 上 BFloat16 GQA 配置的显著加速比，说明现有 VEC 基线无法充分利用 RVV 指令集的并行能力。

实现拆解

1. 新增 RVV 微内核 (csrc/cpu/cpu_attn_rvv.hpp) - 定义 VLEN=128 硬编码的固定宽度 typedef（fixed_vfloat32m2_t 等）。 - 实现 load_row8_B_as_f32 特化模板：对 float16/bfloat16 分别通过 Zvfh 硬指令或标量回退加载并转换为 FP32。 - 核心模板 gemm_micro_rvv_fma_Mx8_Ku4：Mx8 tile，内层 K 展开 4，使用 __riscv_vfmacc_vf 完成标量广播乘累加，寄存器压力可控（M=8 仅用 18 个寄存器）。
2. ISA 枚举与分发集成 (csrc/cpu/generate_cpu_attn_dispatch.py) - 在 ISA_TYPES 字典中新增 "RVV": 5，原有 VSX 后移至 6。 - 在 ISA_FOR_32 列表中加入 "RVV" 使其支持 head_dim 32 对齐场景。 - 生成 dispatch 宏头文件时，为 RISC-V 架构生成两个预处理器分支：
   - __riscv_v_min_vlen == 128 时包含 cpu_attn_rvv.hpp 并生成 RVV+VEC+VEC16 的 case。
   - 其他 RISC-V 只含 VEC/VEC16，安全降级。
3. Python 调度侧集成 (vllm/v1/attention/backends/cpu_attn.py) - 新增 _riscv_supports_rvv_vlen128() 函数：通过读 /proc/cpuinfo 确认只有 zvl128b 且无更高 VLEN 标志，精确匹配 CMake 编译路径。 - 在 _get_attn_isa() 中检测 CpuArchEnum.RISCV 且 _riscv_supports_rvv_vlen128() 时返回 "rvv"。 - 将 CpuArchEnum.RISCV 加入 _CPU_ARCH_PREFER_MIXED_BATCH 元组，确保 RISC-V 也使用混合 batch 调度策略。

4. 类型系统与 Bug 修复 (csrc/cpu/cpu_types_riscv_impl.hpp) - 添加 FP8 空标签结构体 fp8_e4m3_tag / fp8_e5m2_tag 及 BF16Vec32 的对应构造函数存根，避免 GCC 15 -Wtemplate-body 在死分支上的名字查找报错。 - 将 RISCV_BF16_SUPPORT 宏替换为编译器预定义的 __riscv_zvfbfmin，避免手工维护 flag 与扩展不一致。 - 修复 FP32Vec8::exp() 和 FP32Vec16::exp()：对输入钳位到 [-87.33, 88.72]，防止多项式逼近计算中出现 -inf * 0.0 = NaN。
5. 测试与基准配套 - tests/kernels/attention/test_cpu_attn.py: 新增 test_varlen_with_paged_kv_normal_rvv，378 个参数组合，仅在 RISC-V 上运行。 - benchmarks/kernels/cpu/benchmark_cpu_attn.py: 重构 get_attn_isa() 统一走 _get_attn_isa，并添加 --isa 参数 rvv 选项。 - cmake/cpu_extension.cmake: 微调 -mrvv-vector-bits=zvl 传递，确保编译与运行时 VLEN 一致。

关键文件：

- csrc/cpu/cpu_attn_rvv.hpp（模块 计算内核；类别 source；类型 core-logic；符号 TileGemmRVV, AttentionImpl）：新核心文件，实现 RVV 微内核（GEMM、加载、存储），决定性能提升。
- vllm/v1/attention/backends/cpu_attn.py（模块 注意力后端；类别 source；类型 dependency-wiring；符号 _riscv_supports_rvv_vlen128）：Python 调度入口，决定是否启用 RVV 路径，并添加运行时检测。
- csrc/cpu/generate_cpu_attn_dispatch.py（模块 代码生成；类别 source；类型 dependency-wiring）：代码生成器，负责将 RVV 纳入 dispatch 枚举和宏定义。
- benchmarks/kernels/cpu/benchmark_cpu_attn.py（模块 基准测试；类别 source；类型 dependency-wiring）：基准测试支持 RVV ISA 选项，方便性能回归。
- tests/kernels/attention/test_cpu_attn.py（模块 注意力测试；类别 test；类型 test-coverage；符号 test_varlen_with_paged_kv_normal_rvv）：新增 RVV 变长 test，378 参数组合验证正确性。
- csrc/cpu/cpu_types_riscv_impl.hpp（模块 计算内核；类别 source；类型 core-logic）：修复 exp NaN、添加 FP8 stubs、改进 BF16 检测，保证编译正确。

关键符号：_riscv_supports_rvv_vlen128, _get_attn_isa, gemm_micro_rvv_fma_Mx8_Ku4, load_row8_B_as_f32, test_varlen_with_paged_kv_normal_rvv

关键源码片段

csrc/cpu/cpu_attn_rvv.hpp

新核心文件，实现 RVV 微内核（GEMM、加载、存储），决定性能提升。

```
// 文件：csrc/cpu/cpu_attn_rvv.hpp（新增）
// RVV 微内核模板：Mx8 tile，K 循环展开 4，使用标量广播 FMA

template <int32_t M, typename kv_cache_t>
FORCE_INLINE void gemm_micro_rvv_fma_Mx8_Ku4(
    const float* __restrict A, // [M x K]
    const kv_cache_t* __restrict B, // [K x 8]
    float* __restrict C, // [M x 8]
```

```
    int64_t lda, int64_t ldb, int64_t ldc, int32_t K, bool accumulate) {
  static_assert(1 <= M && M <= 8, "M must be in [1,8]");
  constexpr size_t vl = 8; // VLEN=128 时 m2 刚好装 8 个 FP32

  // 累加寄存器数组 (m2 LMUL, 8 个 FP32)
  fixed_vfloat32m2_t c_reg[M];

  if (!accumulate) {
    // 清零初始化
    for (int i = 0; i < M; ++i)
      c_reg[i] = __riscv_vfmv_v_f_f32m2(0.0f, vl);
  } else {
    // 从 C 矩阵加载当前值
    for (int i = 0; i < M; ++i)
      c_reg[i] = __riscv_vle32_v_f32m2(C + i * ldc, vl);
  }

  // K 主循环，每次处理 4 列（展开），减少掩码开销
  for (int32_t k = 0; k < K; k += 4) {
    //……实际实现省略冗余代码，结构同上 ……
    // 每个 M 行执行 4 次 vfmacc_vf：标量 * 向量 + 累加器
  }

  // 写回 C 矩阵
  for (int i = 0; i < M; ++i)
    __riscv_vse32_v_f32m2(C + i * ldc, c_reg[i], vl);
}
```

vllm/v1/attention/backends/cpu_attn.py

Python 调度入口，决定是否启用 RVV 路径，并添加运行时检测。

```
# 文件：vllm/v1/attention/backends/cpu_attn.py（片段）
import functools

@functools.lru_cache(maxsize=1)
def _riscv_supports_rvv_vlen128() -> bool:
    """检测当前主机是否编译了 VLEN=128 的 RVV 路径。

    读取 /proc/cpuinfo 确认只有 zvl128b 且无更高 VLEN 标志，
    与 CMake 编译时自动检测逻辑一致，避免 Python 请求不存在的 ISA。
    """
    try:
        with open("/proc/cpuinfo") as f:
            cpuinfo = f.read()
    except OSError:
        return False
    if "zvl128b" not in cpuinfo:
        return False
    # 确保没有更高 VLEN 标志（否则 CMake 会跳过 RVV-128 编译）
```

```
    return all(f"zvl{n}b" not in cpuinfo for n in (256, 512, 1024))


def _get_attn_isa(dtype, block_size, head_size=None, kv_cache_dtype=None):
    # … 已有逻辑 …
    supports_riscv = arch == CpuArchEnum.RISCV
    # …
    if supports_riscv and _riscv_supports_rvv_vlen128():
        return "rvv"
    # … 其他 ISA …
```

csrc/cpu/generate_cpu_attn_dispatch.py

代码生成器，负责将 RVV 纳入 dispatch 枚举和宏定义。

```
# 文件：csrc/cpu/generate_cpu_attn_dispatch.py（片段）

# ISA 编码表中插入 RVV (5)，原 VSX 后移
ISA_TYPES = {
    "AMX": 0,
    "VEC": 1,
    "VEC16": 2,
    "NEON": 3,
    "VXE": 4,
    "RVV": 5, # 新增
    "VSX": 6, # 原 5 改 6
}

# 支持 head_dim 32 对齐的 ISA 列表中加入 RVV
ISA_FOR_32 = ["AMX", "NEON", "VEC", "VEC16", "VXE", "RVV", "VSX"]

# 生成头文件时，为 RISC-V 创建两个预处理器分支
def generate_header_file():
    # … 已有其他架构的头文件包含 …

    // cpu_attn_rvv.hpp 仅在 VLEN==128 时包含
    #if defined(__riscv) && defined(__riscv_v_min_vlen) && \
        __riscv_v_min_vlen == 128
      #include "cpu_attn_rvv.hpp"
    #endif

    # 生成 dispatch 宏：针对 VLEN==128 和 其他 RISC-V 分别生成
    header += _macro_block(
        "#elif defined(__riscv) && defined(__riscv_v_min_vlen) "
        "&& __riscv_v_min_vlen == 128",
        ["RVV", "VEC", "VEC16"], # 包含 RVV 路径
        fp8=False,
    )
    header += _macro_block(
        "#elif defined(__riscv)",
```

```
        ["VEC", "VEC16"], # 无 RVV 路径，安全降级
        fp8=False,
    )
```

评论区精华

1. VLEN 硬编码与 Zvfh 检查（reviewer: camel-cdr） - 指出 riscv_rvv_vector_bits(128) 仅在 VLEN=128 时合法，且未检查 Zvfh 支持。作者在 fb87b32 中修复：添加 #if defined(__riscv_v_min_vlen) && __riscv_v_min_vlen == 128 门控，并在 _riscv_supports_rvv_vlen128() 中只允许 zvl128b 最高。Zvfh 检查通过 #ifdef __riscv_zvfh 实现运行时条件加载。
2. Key Cache 转置优化建议（gemini-code-assist） - 建议使用 strided vector stores（vsse32.v / vsse16.v）替代当前标量循环转置。作者在讨论中表示将在后续 PR 中优化（当前已合并）。
3. 预提交与合并冲突修复（作者自述） - 经历多次 main 合并解决冲突，特别是与 #40451 (Power VSX) 的枚举值冲突和 vllm/v1/attention/backends/cpu_attn.py 中的装饰器冲突。

- VLEN 硬编码与 Zvfh 检查缺失 (correctness): 已修复：通过预处理器门控限制 RVV 路径只在 VLEN=128 时编译，运行时 Python 检测进一步匹配。
- Key cache 转置优化建议 (performance): 讨论中同意该优化，但未在当前 PR 中实现，推至后续 PR。
- 合并冲突与预提交修复 (other): 已解决，所有冲突正确合并。

风险与影响

- 风险：
  - VLEN 硬编码 128：二进制与 VLEN 严格绑定，若部署到 VLEN=256/512 的设备上会回退到 VEC/VEC16，性能损失可接受但未告警。计划将来使用 RVVI 宏支持可变 VLEN 已在 TODO 中。
  - 平台检测依赖 /proc/cpuinfo：在容器 / 非 Linux 环境可能无法正确检测，但 _riscv_supports_rvv_vlen128 返回 False 时会降级到 VEC/VEC16，功能不受影响。
  - FP8 KV cache 不支持：RISC-V 后端未实现 FP8 路径，dispatch 不会为 RISC-V 生成 FP8 case，调用时会走 TORCH_CHECK 报错。但主流场景仍以 auto 为主，影响面小。
  - 新增测试仅在 RISC-V 上执行：CI 中若未配置 RISC-V 运行器，该测试将被跳过，可能遗漏回归。但 x86 等不相关。
  - 影响：用户侧：RISC-V 平台用户可获得 2-4 倍注意力性能提升，且无需额外配置。非 RISC-V 用户无影响。 系统侧：扩展了 vLLM 的 CPU 后端 ISA 调度层，新增 CpuArchEnum.RISCV 枚举和向量类型兼容性层（FP8 stubs）。 团队侧：需要维护 RVV 内核代码及 CI 基础设施（若有 RISC-V runner）。设计上的 VLEN 抽象化值得后续投入。
- 风险标记：VLEN 硬编码限制，平台检测依赖 /proc/cpuinfo, FP8 KV cache 不支持 RISC-V, 测试仅在 RISC-V 物理机运行

关联脉络

- PR #41912 [CPU][RISC-V] Add fp8 / FP32Vec16 stubs to satisfy GCC 15 -Wtemplate-body: 作者从本 PR 拆出的前置修复，解决 GCC 15 模板编译问题，本 PR 依赖它。
- PR #40451 [CPU] Power VSX Attention Backend: 与本 PR 有冲突（ISA 枚举值冲突、cpu_attn.py 中的装饰器冲突），需在合并时协调。
- PR #41913 [CPU][RISC-V] Fix platform detection and exp NaN: 作者提到拆出的另一个小 PR（可能），与本 PR 的修复部分重叠。