PR #38112 完整报告

vllm-project/vllm

[CPU] Added faster exp routine for lower precision data types.

合并时间：2026-04-23 21:14

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/38112

执行摘要

在 ARM CPU NEON 平台上，为 BF16/FP16 类型的注意力 softmax 计算引入了更快的指数函数近似，通过三阶多项式替代完整 exp，在保证低精度下的 1ULP 精度的同时，实现 3-4% 的端到端加速。所有分派均在编译期完成，不引入运行时开销。

功能与动机

低精度数据类型（BF16/FP16）在注意力 softmax 中广泛使用，但标准 exp 函数计算开销较大。该项目观察到在 neoverse V1 上，将 exp 替换为针对低精度调优的近似后，注意力计算可在不影响模型精度的情况下提升 3-4% 性能。

实现拆解

1. 定义快速 exp 近似（csrc/cpu/cpu_arch_macros.h）：新增 fast_exp_f16 函数，它基于 Arm 优化库的 expf 算法，但使用三阶多项式 exp(r) ≈ 1 + r + r²*(c3 + c2*r)，精度对 FP16/BF16 为 1ULP，输入范围限制在 [-87.683, 88.376] 之外时饱和。
2. 集成到注意力 softmax（csrc/cpu/cpu_attn_impl.hpp）：在 apply_softmax 和 apply_softcap 中，通过编译期常量 IsReducedPrecision 判断 query 类型，再结合 #ifdef __aarch64__ 宏，在 ARM 上为低精度类型调用 fast_exp_f16，否则使用原 fast_exp 或 std::exp。
3. 避免 ISA 特异性污染：所有平台宏仅在 cpu_attn_impl.hpp 中使用，保持了 cpu_arch_macros.h 中宏的通用性，并添加了详细注释说明设计意图。
4. 兼容性处理：x86 平台不受影响，因为 #ifdef __aarch64__ 确保 fast_exp_f16 仅在 ARM 编译。

关键代码片段（csrc/cpu/cpu_attn_impl.hpp 中 exp 计算部分）：

csrc/cpu/cpu_arch_macros.h

定义了 fast_exp_f16 函数，是性能优化的核心实现。

csrc/cpu/cpu_attn_impl.hpp

集成了 fast_exp_f16，在 softmax/softcap 中根据类型和平台选择 exp 实现。

关键源码片段

csrc/cpu/cpu_arch_macros.h

定义了 fast_exp_f16 函数，是性能优化的核心实现。

```
// 在 DEFINE_FAST_EXP 宏中新增 fast_exp_f16
// 基于 Arm 优化库 expf AdvSIMD，但使用更低阶多项式
auto neon_expf_f16 = [&](float32x4_t values) __attribute__((always_inline)) {
    // 输入范围限制：[-87.683, 88.376] 外饱和
    const uint32x4_t lt_lower = vcltq_f32(values, lower_bound);
    const uint32x4_t gt_upper = vcgtq_f32(values, upper_bound);
    float32x4_t n = vrndaq_f32(vmulq_f32(values, inv_ln2));
    float32x4_t r = vfmsq_n_f32(values, n, ln2);
    uint32x4_t e = vshlq_n_u32(vreinterpretq_u32_s32(vcvtq_s32_f32(n)), 23);
    float32x4_t r2 = vmulq_f32(r, r);
    // exp(r) ≈ 1 + r + r^2*(c3 + c2*r)，三阶多项式
    float32x4_t q = vfmaq_n_f32(f_c3, r, f_c2);
    float32x4_t s = vaddq_f32(vdupq_n_f32(1.0f), r);
    float32x4_t p = vfmaq_f32(s, q, r2);
    float32x4_t y = vreinterpretq_f32_u32(vaddq_u32(vreinterpretq_u32_f32(p), e));
    y = vbslq_f32(lt_lower, vdupq_n_f32(0.0f), y);
    y = vbslq_f32(gt_upper, vdupq_n_f32(INFINITY), y);
    return y;
};
// 包装为 FP32Vec16 接口，处理四个 128 位向量
auto fast_exp_f16 = [&](const vec_op::FP32Vec16& vec) __attribute__((always_inline)) {
    float32x4x4_t result;
    result.val[0] = neon_expf_f16(vec.reg.val[0]);
    result.val[1] = neon_expf_f16(vec.reg.val[1]);
    result.val[2] = neon_expf_f16(vec.reg.val[2]);
    result.val[3] = neon_expf_f16(vec.reg.val[3]);
    return vec_op::FP32Vec16(result);
};
```

csrc/cpu/cpu_attn_impl.hpp

集成了 fast_exp_f16，在 softmax/softcap 中根据类型和平台选择 exp 实现。

```
// 在 apply_softmax 函数中，exp 计算部分
#if defined(DEFINE_FAST_EXP)
    // 编译期判断是否低精度类型（BF16/Half）
    bool constexpr IsReducedPrecision =
        std::is_same_v<query_t, c10::BFloat16> ||
        std::is_same_v<query_t, c10::Half>;
    // 仅 ARM 平台使用 fast_exp_f16，否则用通用 fast_exp
    #ifdef __aarch64__
    if constexpr (IsReducedPrecision) {
        vec = fast_exp_f16(vec);
    } else
    #endif
    {
        vec = fast_exp(vec);
    }
    // 保存为 prob_buffer_t
```

```
    prob_buffer_vec_t output_vec(vec);
    output_vec.save(curr_prob_buffer_iter);
#else
    // 不使用快速 exp 时，调用 std::exp
    vec.save(curr_logits_buffer_iter);
    for (int32_t k = 0; k < 16; ++k) {
        curr_logits_buffer_iter[k] = std::exp(curr_logits_buffer_iter[k]);
    }
    vec = vec_op::FP32Vec16(curr_logits_buffer_iter);
#endif
```

评论区精华

- fadara01要求将运行时分支改为编译期分派："can we avoid having lowp as a runtime branch? we should know at compile time"——最终采用 if constexpr + #ifdef 方案。
- bigPYJ1151提议将分派移到宏内部，但 almayne指出宏作用域限制："fast_exp is also used in cpu_fused_moe.cpp. Neither query_t nor q_buffer_t are used there."——最终决定在 cpu_attn_impl.hpp 中处理。
- fadara01支持原始版本（在宏中定义别名）以避免 ISA ifdef，但 bigPYJ1151认为显式平台检查更清晰："The explicit platform check may help to remind."——最终采用平台宏方案。

风险与影响

- 精度风险：fast_exp_f16 仅对 BF16/FP16 保证 1ULP，若未来扩展至其他类型需重新验证。
- 平台限制：优化仅作用于 ARM NEON，x86 保持不变。
- 测试覆盖：未引入新测试用例，仅依赖现有注意力基准测试。
- 影响范围：所有使用 CPU BF16/FP16 注意力计算的模型受益，尤其 ARM 平台。

关联脉络

此 PR 与历史 PR #39789（XPU 融合禁用）类似，均为平台特定优化，但本 PR 更关注通用性与特殊优化的分离设计。后续可能需要为 x86 添加等效的低精度 exp 优化，或在 MoE 中评估使用该近似的可行性。