

PR #48577 完整报告

vllm-project/vllm

[CPU][Spec Decode] Optimize GDN conv path for speculative decoding

合并时间: 2026-07-27 14:20

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/48577>

执行摘要

PR #48577 优化了 CPU 上 Qwen3.5 GDN 推测解码的卷积路径，通过引入 AMX C++ 内核替代逐序列的 torch conv1d，在 MTP 场景下获得约 18% 的吞吐提升，且准确率无损。核心在于新增的 `causal_conv1d_update_multi_kernel_impl` 函数和 Python 层推理路径的重构。

功能与动机

根据 PR body，目的是“Improve CPU serving performance for Qwen3.5 GDN speculative decoding with MTP by replacing fallback torch conv1d path with cpp implementation when AMX is available”。不使用 AMX 时自动回退到 PyTorch 实现，确保兼容性。

实现拆解

1. 新增批量卷积内核：在 `csrc/cpu/sgl-kernels/conv.cpp` 中新增 `causal_conv1d_update_multi_kernel_impl` 模板函数，接受 `num_accepted_tokens` 数组，允许每个序列具有不同的已接受 draft token 数量。函数使用 `tinygemm_kernel` 计算卷积，并通过 `memmove/memcpy` 高效滚动状态缓冲区。
2. 更新 C++/Python 绑定：修改 `csrc/cpu/torch_bindings.cpp` 和 `vllm/_custom_ops.py`，将 `causal_conv1d_update_cpu` 的 `cache_seqLens` 参数替换为 `num_accepted_tokens`，使新内核可被 Python 层调用。
3. 重构 Python 推理路径：在 `vllm/model_executor/layers/mamba/ops/cpu/gdn_attention.py` 中重写 `_spec_forward` 函数。根据 `torch.cpu._is_amx_tile_supported()` 判断是否可用本地内核：若可用则直接调用 `ops.causal_conv1d_update_cpu` 批量处理整个批次；否则回退到逐序列的 PyTorch `F.conv1d`。同时移除多处多余的 `.contiguous()` 调用。
4. 消除冗余数据移动：在 `_cpu_gdn_attention_spec_aware` 中，将 `mixed_qkv.contiguous()`、`b.contiguous()`、`a.contiguous()` 延迟到非 spec 路径，避免过早拷贝；在 `_cpu_gdn_attention_nonspec` 中移除不必要的 `state_indices_tensor.contiguous()`。
5. 扩充测试套件：在 `tests/kernels/mamba/cpu/test_cpu_gdn_ops.py` 中新增 `test_spec_aware_mixed_routing_preserves_token_order`、`test_spec_aware_nonspec_materializes_state_indices` 等多个测试用例，验证 spec 感知路由下 token 顺序、状态索引连续性，以及两阶段调用与单阶段调用的数值一致性。

`csrc/cpu/sgl-kernels/conv.cpp`

新增批量卷积前向模板函数，支持多个 accepted token 同时计算，并优化状态滚动，是性能提升的核心。

```

// causal_conv1d_update_multi_kernel_impl – 批量处理多 draft token 的卷积前向和状态更新
template <typename scalar_t>
void causal_conv1d_update_multi_kernel_impl(
    scalar_t* __restrict__ out,
    const scalar_t* __restrict__ input,
    scalar_t* __restrict__ conv_states,
    const scalar_t* __restrict__ weight,
    const scalar_t* __restrict__ bias,
    const int32_t* __restrict__ num_accepted_tokens, // 每个序列已接受的 draft 数
    const int32_t* __restrict__ conv_indices,
    bool silu_activation,
    int64_t batch, int64_t dim, int64_t seqlen, int64_t width,
    int64_t state_len, int64_t conv_state_slot_stride) {
    constexpr int64_t BLOCK_N = block_size_n() * 2;
    const int64_t NB = div_up(dim, BLOCK_N);
    AT_DISPATCH_BOOL2(bias != nullptr, has_bias, silu_activation, has_silu, [&] {
        at::parallel_for(0, batch * NB, 0, [&](int64_t begin, int64_t end) {
            int64_t bs, nb;
            data_index_init(begin, bs, batch, nb, NB);
            for (int64_t i = begin; i < end; ++i) {
                const int64_t nb_start = nb * BLOCK_N;
                const int64_t nb_size = std::min(dim - nb_start, BLOCK_N);
                const int32_t conv_state_index = conv_indices[bs];
                const int32_t history_offset = num_accepted_tokens[bs] - 1;
                switch (width << 4 | nb_size >> 4) {
                    case 0x42:
                        tinygemm_kernel<scalar_t, 4, 32, has_bias, has_silu>::apply(
                            input + bs * seqlen * dim + nb_start,
                            weight + nb_start * width,
                            out + bs * seqlen * dim + nb_start,
                            has_bias ? bias + nb_start : nullptr,
                            conv_states + conv_state_index * conv_state_slot_stride +
                                history_offset * dim + nb_start,
                            true, seqlen, dim, true);
                        break;
                    case 0x44:
                        tinygemm_kernel<scalar_t, 4, 64, has_bias, has_silu>::apply(
                            input + bs * seqlen * dim + nb_start,
                            weight + nb_start * width,
                            out + bs * seqlen * dim + nb_start,
                            has_bias ? bias + nb_start : nullptr,
                            conv_states + conv_state_index * conv_state_slot_stride +
                                history_offset * dim + nb_start,
                            true, seqlen, dim, true);
                        break;
                    default:
                        TORCH_CHECK(false, "Unexpected block size, ", width, " x ", nb_size);
                }
            }
            data_index_step(bs, batch, nb, NB);
        }
    });
}

```

```

    }
  });
});
at::parallel_for(0, batch, 0, [&](int64_t begin, int64_t end) {
  for (int64_t bs = begin; bs < end; ++bs) {
    const int32_t conv_state_index = conv_indices[bs];
    const int32_t num_accepted = num_accepted_tokens[bs];
    scalar_t* state = conv_states + conv_state_index * conv_state_slot_stride;
    std::memmove(state, state + num_accepted * dim,
                  (state_len - seqlen) * dim * sizeof(scalar_t));
    std::memcpy(state + (state_len - seqlen) * dim,
                 input + bs * seqlen * dim,
                 seqlen * dim * sizeof(scalar_t));
  }
});
}

```

评论区精华

bigPYJ1151: 对于 `non_spec` 路径的 `state_indices_tensor`，它是来自 `non_spec_query_start_loc` 的原始 1D 连续 tensor，不需要 `.contiguous()`。tianmu-li 最终移除。

bigPYJ1151: 将 `num_acc_cpu` 的转换移到 AMX 路径外避免冗余。tianmu-li 处理后将转换移除。

bigPYJ1151: `mixed_qkv_spec` 由 `index_select` 生成已连续，无需 `reshape`。tianmu-li 改用 `view`。

fadara01: 是否可使用 `causal_conv1d.py` 中的接口减少重复？tianmu-li 表示现有 helper 不支持，并已创建 follow-up #49650 专门扩展该模块。

风险与影响

- 性能风险：新 AMX 内核依赖硬件特性，非 AMX CPU 自动回退到 PyTorch 路径，有单元测试覆盖回退正确性。
- 正确性风险：状态滚动逻辑（`memmove/memcpy`）边界敏感，但两侧对比测试（单次 vs 分两次）覆盖了关键路径。
- 兼容性风险：op 签名参数更名（`cache_seqLens` → `num_accepted_tokens`）仅影响内部调用，已全部更新。
- 影响范围：仅 CPU 推测解码（MTP）用户受益，吞吐提升约 18%；非 AMX 平台无变化。

关联脉络

本 PR 与 #49910（显式关闭推测解码 runner）同属推测解码改进方向。讨论中提出的 follow-up #49650 将进一步把 conv 操作统一到 `causal_conv1d.py`，减少 python 端的代码重复，值得后续关注。此外，同仓库中的 #49736（mamba_hybrid GPU-CPU 同步修复）也涉及 Mamba 家族模型，展示了跨平台统一注意力的趋势。