PR #45391 完整报告

vllm-project/vllm

[CPU] Refine CPU attention frontend

合并时间：2026-06-15 10:26

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/45391

执行摘要

- 一句话：CPU 注意力前端重构：移除 SDPA，ISA 检查提前，新增 head_dim 48
- 推荐动作：值得精读。该 PR 展示了 CPU 注意力后端的演进方向：统一内核、移除对 PyTorch SDPA 的依赖。对于关注 CPU 推理性能的开发者有参考价值，特别是其中的基准测试方法值得借鉴。

功能与动机

减少注意力前端的复杂性，移除过时的 SDPA 路径，将 ISA 检查从热路径移出以避免重复查询，并扩展对 head_dim 48 的支持以覆盖更多模型。同时为编码器注意力提供统一的内核实现替代 SDPA，提升变长输入场景的性能。

实现拆解

1. 移除 SDPA 预填充路径：在 CPUAttentionMetadataBuilder.__init__ 中删除 use_sdpa_prefill 和相关逻辑，不再需要 split_decodes_and_prefills 工具函数。现在所有注意力类型都使用统一的 cpu_attention_with_kv_cache 内核。
2. 将 ISA 检查移到初始化：确保 self.isa 在 builder 初始化时计算一次，而不是在每次 forward 时重新获取。这减少了查询 ISA 的开销。
3. 添加 head_dim 48 支持：在 generate_cpu_attn_dispatch.py 中将 HEAD_DIMS_16 列表中添加 48，使 VEC16 ISA 能够处理 head_dim 48 的注意力计算。
4. 添加编码器注意力支持：在 CPUAttentionMetadata 中添加 encoder_cache 字段；在 builder 的 build 方法中根据 EncoderOnlyAttentionSpec 分配临时 KV 缓存（基于 block table）。在 forward 中，编码器注意力路径现在直接调用 cpu_attention_with_kv_cache，替代原来的 _run_sdpa_forward。
5. 更新测试：在 test_cpu_attn.py 中添加 ref_varlen_encoder_attn 参考实现和 varlen_encoder_attention 测试函数，覆盖 VEC 和 AMX 两个 ISA 的编码器注意力。

关键文件：

- vllm/v1/attention/backends/cpu_attn.py（模块 注意力后端；类别 source；类型 core-logic；符号 _run_sdpa_forward, _make_alibi_bias, _make_sliding_window_bias）：核心变更：移除 SDPA、添加编码器注意力支持、ISA 检查提前、新增 encoder_cache 字段

- tests/kernels/attention/test_cpu_attn.py（模块 CPU 注意力测试；类别 test；类型 test-coverage；符号 tensor_cache, ref_varlen_encoder_attn, varlen_encoder_attention, test_varlen_encoder_attention_vec）：新增编码器注意力参考实现和测试函数，覆盖 VEC 和 AMX ISA
- csrc/cpu/cpu_attn_impl.hpp（模块 C++ 内核；类别 source；类型 core-logic）：C++ 端调整：添加 kv_end_pos 参数用于正确计算 mask 边界
- csrc/cpu/generate_cpu_attn_dispatch.py（模块 代码生成；类别 source；类型 core-logic）：添加 head_dim 48 到 VEC16 支持列表，新增 48 头维度的 kernel 生成

关键符号：_run_sdpa_forward, _make_alibi_bias, _make_sliding_window_bias, ref_varlen_encoder_attn, varlen_encoder_attention, tensor_cache

关键源码片段

vllm/v1/attention/backends/cpu_attn.py

核心变更：移除 SDPA、添加编码器注意力支持、ISA 检查提前、新增 encoder_cache 字段

```
@staticmethod
def __init__(
    self,
    kv_cache_spec: AttentionSpec,
    layer_names: list[str],
    vllm_config: VllmConfig,
    device: torch.device,
) -> None:
    super().__init__(kv_cache_spec, layer_names, vllm_config, device)

    self.kv_cache_spec = kv_cache_spec
    self.vllm_config = vllm_config

    parallel_config = vllm_config.parallel_config
    self.num_kv_heads = vllm_config.model_config.get_num_kv_heads(parallel_config)
    self.num_heads = vllm_config.model_config.get_num_attention_heads(
        parallel_config
    )
    self.head_dim = kv_cache_spec.head_size
    self.dtype = vllm_config.model_config.dtype
    self.window_size = getattr(kv_cache_spec, "sliding_window", -1)
    if self.window_size is None:
        self.window_size = -1
    self.block_size = vllm_config.cache_config.block_size
    kv_cache_dtype_str = vllm_config.cache_config.cache_dtype

    # ISA 检查在初始化时完成，避免重复查询
    self.isa = _get_attn_isa(
        self.dtype,
        self.block_size,
        self.head_dim,
```

```
            kv_cache_dtype_str,
        )
        self.is_cross_attention = isinstance(kv_cache_spec, CrossAttentionSpec)
        # 新增：识别 encoder-only attention 类型
        self.is_encoder_only_attention = isinstance(
            kv_cache_spec, EncoderOnlyAttentionSpec
        )
```

tests/kernels/attention/test_cpu_attn.py

新增编码器注意力参考实现和测试函数，覆盖 VEC 和 AMX ISA

```
@torch.inference_mode()
def ref_varlen_encoder_attn(
    query: torch.Tensor, # [token, q_head_num, head_dim]
    key: torch.Tensor, # [token, kv_head_num, head_dim]
    value: torch.Tensor,
    seq_lens: list[int],
    scale: float,
    sliding_window: int | None = None,
) -> torch.Tensor:
    num_seqs = len(seq_lens)
    dtype = query.dtype
    output = torch.empty_like(query)
    start_idx = 0
    for i in range(num_seqs):
        seq_len = seq_lens[i]
        q = query[start_idx : start_idx + seq_len].float()
        k = key[start_idx : start_idx + seq_len].float()
        v = value[start_idx : start_idx + seq_len].float()
        q *= scale

        # 处理 GQA：key/value 可能 head 数少于 query
        if q.shape[1] != k.shape[1]:
            k = torch.repeat_interleave(k, q.shape[1] // k.shape[1], dim=1)
            v = torch.repeat_interleave(v, q.shape[1] // v.shape[1], dim=1)

        attn = torch.einsum("qhd,khd->hqk", q, k).float()
        empty_mask = torch.ones(seq_len, seq_len)
        if sliding_window is not None:
            mask = (
                torch.triu(empty_mask, diagonal=1 - sliding_window).bool()
                ^ torch.triu(empty_mask, diagonal=sliding_window).bool()
            ).logical_not()
        else:
            mask = empty_mask.logical_not()

        attn.masked_fill_(mask, float("-inf"))
        attn = torch.softmax(attn, dim=-1)
        out = torch.einsum("hqk,khd->qhd", attn, v).to(dtype=dtype)
```

```
        output[start_idx : start_idx + seq_len].copy_(out)

        start_idx += seq_len

    return output
```

评论区精华

fadara01 质疑新实现是否在非 x86 架构上与 SDPA 一样快。bigPYJ1151 认为 SDPA 对变长输入使用 Python 循环，而新内核一次处理所有 token，应该更快。almayne 使用 Whisper 测试未发现回归，但使用 BGE-small 嵌入模型（纯编码器）测试显示请求吞吐量和 token 吞吐量均提升约 37%，证明新实现的有效性。

- 替换 SDPA 对编码器注意力的性能影响 (performance): 新实现性能优于 SDPA，尤其是在变长输入场景。后续可考虑替换 ViT 注意力以进一步优化。

风险与影响

- 风险：风险较低。主要风险在于移除 SDPA 后，若新内核在某些 CPU 架构上性能下降，但测试显示在 x86 上至少持平；非 x86 架构（如 ARM）未专门测试，但由 _get_attn_isa 根据 ISA 自动选择实现。新增的编码器注意力路径经过单元测试覆盖，但可能遗漏边界情况（如 sliding window 与 encoder 结合使用）。ISA 检查提前到初始化可能无法应对运行时 CPU 特性变化（罕见）。
- 影响：影响范围限于 CPU 后端的注意力计算，对 GPU 等其他后端无影响。用户无需更改代码即可受益于性能提升，特别是使用编码器模型（如 Whisper、嵌入模型）的用户。移除 SDPA 依赖也降低了维护成本。
- 风险标记：移除 SDPA 回退路径，新增编码器注意力路径，非 x86 架构未充分测试

关联脉络

- 暂无明显关联 PR