PR #42707 完整报告

vllm-project/vllm

[CPU] Add fused GDN support for AMX CPU platform

合并时间：2026-05-18 18:04

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/42707

执行摘要

- 一句话：为 AMX CPU 添加融合 GDN 算子与因果卷积内核
- 推荐动作：值得精读以学习 AMX 算子集成模式（Python 绑定 + C++ 注册 + 平台检测），但合并前引入的严重 review 意见未解决，建议在实际部署前对 4 个风险点进行二次修复并补充单元测试。

功能与动机

在 CPU 上高效运行 Qwen3.5 等采用 GDN 注意力机制的模型，需要利用 AMX 指令集实现融合计算，减少内存带宽开销。AMX 平台对张量布局有特殊要求（如 conv 状态需 SD 布局），本 PR 通过新增定制算子和运行时检测来激活该加速路径。

实现拆解

1. Python 算子接口（vllm/_custom_ops.py）：新增 chunk_gated_delta_rule_cpu、fused_sigmoid_gating_delta_rule_update_cpu、fused_gdn_gating_cpu 三个 GDN 相关函数，以及 causal_conv1d_weight_pack、causal_conv1d_fwd_cpu、causal_conv1d_update_cpu 三个因果卷积函数，统一通过 torch.ops._C 调用底层 C++ 实现。
2. C++ 算子注册（csrc/cpu/torch_bindings.cpp）：声明并实现上述算子的 C++ 签名，通过 TORCH_LIBRARY_EXPAND 注册到 PyTorch 的自定义算子库，绑定后端 SGL 内核。
3. AMX 专用 GDN 注意力核心（vllm/model_executor/layers/mamba/ops/cpu/gdn_attention.py）：新增 cpu_gdn_attention_core_amx 函数，在检测到 AMX 支持时跳转，该函数使用 causal_conv1d_update_cpu 和 fused_sigmoid_gating_delta_rule_update_cpu 实现融合的 decode 与 prefill 路径，并显式处理 conv 状态布局转换。
4. 权重分发适配（vllm/model_executor/layers/utils.py）：在 dispatch_cpu_unquantized_gemm 中增加对非 2D 权重（如 causal conv1d 3D 权重）的判断，在 AMX 平台上调用 causal_conv1d_weight_pack 进行预打包。
5. 平台配置调整（vllm/platforms/cpu.py）：在 check_and_update_config 中检测 AMX 支持且模型包含线性注意力层时，自动禁用前缀缓存和分块预填充（因 AMX GDN 尚不支持这些特性）；同时设置环境变量 VLLM_SSM_CONV_STATE_LAYOUT=SD 以便高效访问 conv 状态。

关键文件：

- vllm/_custom_ops.py（模块 算子接口；类别 source；类型 core-logic；符号 chunk_gated_delta_rule_cpu, fused_sigmoid_gating_delta_rule_update_cpu, fused_gdn_gating_cpu, causal_conv1d_weight_pack）：新增 6 个自定义算子 Python 绑定，是上游调用的入口，对理解整个实现架构最关键。
- csrc/cpu/torch_bindings.cpp（模块 算子绑定；类别 source；类型 core-logic；符号 chunk_gated_delta_rule_cpu, fused_sigmoid_gating_delta_rule_update_cpu, fused_gdn_gating_cpu, causal_conv1d_weight_pack）：C++ 算子注册入口，声明并绑定所有新算子的实现，是 Python 与 C++ 内核的桥梁。
- vllm/model_executor/layers/mamba/ops/cpu/gdn_attention.py（模块 GDN 注意力；类别 infra；类型 infrastructure；符号 cpu_gdn_attention_core_amx）：AMX 专用 GDN 注意力核心函数 cpu_gdn_attention_core_amx，是实际加速逻辑所在。
- vllm/model_executor/layers/utils.py（模块 权重重分发；类别 source；类型 data-contract）：修改 dispatch 逻辑，支持非 2D 权重预打包（causal conv1d），是模型加载适配关键。
- vllm/platforms/cpu.py（模块 平台配置；类别 source；类型 core-logic）：平台配置检测 AMX 并禁用不兼容特性，影响 CPU 推理行为。
- csrc/cpu/sgl-kernels/conv.cpp（模块 卷积内核；类别 source；类型 core-logic）：因果卷积 C++ 实现，修改 conv_state 地址计算以兼容 vLLM KV cache 的 slot stride 布局。
- vllm/model_executor/layers/linear.py（模块 线性层；类别 source；类型 data-contract）：删除 3 行代码，移除旧的条件判断，为新的 dispatch 路径让路。
- csrc/cpu/sgl-kernels/fla.cpp（模块 注意力内核；类别 source；类型 core-logic）：小幅修改以适应更新后的 conv_state 布局参数传递。

关键符号：chunk_gated_delta_rule_cpu, fused_sigmoid_gating_delta_rule_update_cpu, fused_gdn_gating_cpu, causal_conv1d_weight_pack, causal_conv1d_fwd_cpu, causal_conv1d_update_cpu, cpu_gdn_attention_core_amx, dispatch_cpu_unquantized_gemm, check_and_update_config

关键源码片段

csrc/cpu/torch_bindings.cpp

C++ 算子注册入口，声明并绑定所有新算子的实现，是 Python 与 C++ 内核的桥梁。

```
// 在文件头部声明新算子的 C++ 接口
// Adapted from sglang: GDN
std::tuple<at::Tensor, at::Tensor> chunk_gated_delta_rule_cpu(
    const at::Tensor& query, const at::Tensor& key, const at::Tensor& value,
    const at::Tensor& g, const at::Tensor& beta,
    const at::Tensor& initial_state, bool output_final_state,
    const at::Tensor& cu_seqlens, bool head_first, bool use_qk_l2norm_in_kernel,
    double eps = 1e-5);

at::Tensor fused_sigmoid_gating_delta_rule_update_cpu(
    const at::Tensor& A_log, const at::Tensor& dt_bias, const at::Tensor& q,
    const at::Tensor& k, const at::Tensor& v, const at::Tensor& a,
```

```
    const at::Tensor& b, at::Tensor& initial_state_source,
    const at::Tensor& initial_state_indices, const at::Tensor& cu_seqlens,
    bool use_qk_l2norm_in_kernel, double softplus_beta = 1.0,
    double softplus_threshold = 20.0);

std::tuple<at::Tensor, at::Tensor> fused_gdn_gating_cpu(
    const at::Tensor& A_log, const at::Tensor& a, const at::Tensor& b,
    const at::Tensor& dt_bias);

// Adapted from sglang: casual_conv1d kernels
at::Tensor causal_conv1d_weight_pack(const at::Tensor& weight);

at::Tensor causal_conv1d_fwd_cpu(
    const at::Tensor& x, const at::Tensor& weight,
    const std::optional<at::Tensor>& bias,
    const std::optional<at::Tensor>& conv_states,
    const std::optional<at::Tensor>& query_start_loc,
    const std::optional<at::Tensor>& cache_indices,
    const std::optional<at::Tensor>& has_initial_state, bool silu_activation,
    int64_t pad_slot_id, bool is_vnni);

at::Tensor causal_conv1d_update_cpu(
    const at::Tensor& x, const at::Tensor& conv_states,
    const at::Tensor& weight, const std::optional<at::Tensor>& bias,
    bool silu_activation, const std::optional<at::Tensor>& cache_seqlens,
    const std::optional<at::Tensor>& conv_state_indices, int64_t pad_slot_id,
    bool is_vnni);

// 在 TORCH_LIBRARY_EXPAND 块中注册
ops.def(
    "chunk_gated_delta_rule_cpu(Tensor query, Tensor key, Tensor value, "
    "Tensor g, Tensor beta, Tensor initial_state, bool output_final_state, "
    "Tensor cu_seqlens, bool head_first, bool use_qk_l2norm_in_kernel, "
    "float eps=1e-5) -> (Tensor, Tensor)");
ops.impl("chunk_gated_delta_rule_cpu", torch::kCPU, &chunk_gated_delta_rule_cpu);

ops.def(
    "fused_sigmoid_gating_delta_rule_update_cpu(Tensor A_log, Tensor dt_bias, "
    "Tensor q, Tensor k, Tensor v, Tensor a, Tensor b, "
    "Tensor(a!) initial_state_source, Tensor initial_state_indices, "
    "Tensor cu_seqlens, bool use_qk_l2norm_in_kernel, "
    "float softplus_beta=1.0, float softplus_threshold=20.0) -> Tensor");
ops.impl("fused_sigmoid_gating_delta_rule_update_cpu", torch::kCPU,
         &fused_sigmoid_gating_delta_rule_update_cpu);

ops.def(
    "fused_gdn_gating_cpu(Tensor A_log, Tensor a, Tensor b, "
    "Tensor dt_bias) -> (Tensor, Tensor)");
ops.impl("fused_gdn_gating_cpu", torch::kCPU, &fused_gdn_gating_cpu);
```

```
  ops.def(
      "causal_conv1d_weight_pack(Tensor weight) -> Tensor");
  ops.impl("causal_conv1d_weight_pack", torch::kCPU, &causal_conv1d_weight_pack);

  // (causal_conv1d_fwd_cpu 和 causal_conv1d_update_cpu 类似注册 )
```

vllm/model_executor/layers/mamba/ops/cpu/gdn_attention.py

AMX 专用 GDN 注意力核心函数 cpu_gdn_attention_core_amx，是实际加速逻辑所在。

```
  def cpu_gdn_attention_core_amx(
      mixed_qkv: torch.Tensor,
      b: torch.Tensor,
      a: torch.Tensor,
      core_attn_out: torch.Tensor,
      attn_metadata_i: GDNAttentionMetadata,
      layer: torch.nn.Module,
  ):
      """
      AMX 加速的 GDN 注意力核心。输入输出接口与原函数一致，但内部使用
      自定义算子实现融合的因果卷积和门控 delta 规则更新。
      要求 conv 状态为 [num_slots, conv_dim, kernel-1] 的 SD 布局。
      """
      # 获取状态索引和查询起始位置
      state_indices_tensor = attn_metadata_i.non_spec_state_indices_tensor
      query_start_loc = attn_metadata_i.non_spec_query_start_loc
      assert state_indices_tensor is not None
      assert query_start_loc is not None

      # conv state: [num_allocated_slots, kernel-1, conv_dim] → 转置为 SD 布局
      conv_state = layer.kv_cache[0]
      if is_conv_state_dim_first():
          raise RuntimeError("AMX GDN attention requires `SD` conv_state layout.")
      # 注意：这里假设 conv_state 已经是 SD 布局，只需转置一次
      conv_state_t = conv_state.transpose(1, 2) # [slots, conv_dim, kernel-1]

      # ssm state: [slots, heads, v_dim, k_dim] → 重排为 [slots, heads, k_dim, v_dim]
      ssm_state = layer.kv_cache[1]
      num_allocated_slots, head_num, v_dim, k_dim = ssm_state.size()
      # !!! 潜在缺陷：此处使用 .view() 而非 .transpose() 会导致维度语义错误，
      # 见 review 讨论。正确做法为 ssm_state.transpose(2,3) 后 contiguous。
      ssm_state = ssm_state.view(
          num_allocated_slots, head_num, k_dim, v_dim)

      # 确保输入连续
      mixed_qkv = mixed_qkv.contiguous()
      a = a.contiguous()
      b = b.contiguous()

      num_decodes = attn_metadata_i.num_decodes
```

```
    num_decode_tokens = attn_metadata_i.num_decode_tokens
    # ... 后续根据 num_decodes > 0 分别处理 decode 和 prefill
    if num_decodes > 0:
        # decode 路径：先因果卷积更新，再 fused_sigmoid_gating_delta_rule_update
        decode_mixed_qkv = ops.causal_conv1d_update_cpu(
            x=decode_mixed_qkv,
            conv_states=conv_state_t,
            weight=layer.conv1d.weight,
            bias=layer.conv1d.bias,
            silu_activation=layer.activation == "silu",
            conv_state_indices=decode_state_indices,
            is_vnni=True,
        )
        query, key, value = layer.rearrange_mixed_qkv(decode_mixed_qkv)
        attn_out = ops.fused_sigmoid_gating_delta_rule_update_cpu(
            A_log=layer.A_log, dt_bias=layer.dt_bias,
            q=query, k=key, v=value, a=decode_a, b=decode_b,
            initial_state_source=ssm_state,
            initial_state_indices=decode_state_indices,
            cu_seqlens=query_start_loc[:num_decodes+1],
            use_qk_l2norm_in_kernel=True,
        )
        # ... 输出写入 core_attn_out
```

vllm/model_executor/layers/utils.py

修改 dispatch 逻辑，支持非 2D 权重预打包（causal conv1d），是模型加载适配关键。

```
def dispatch_cpu_unquantized_gemm(
    layer: torch.nn.Module,
    remove_weight: bool,
) -> None:
    # ... 原有逻辑 ...

    # 新增：如果权重不是 2D（例如 causal conv1d 的 3D 权重）
    if layer.weight.ndim != 2:
        # 检查是否为 AMX 平台（仅当 AMX 支持时预打包）
        if torch.cpu._is_amx_tile_supported():
            # 将权重 reshape 为 (out_dim, kernel_width) 后调用 weight_pack
            # 注意：原始权重形状为 (dim, kernel_size, conv_dim) 或类似
            # 这里假设 ndim=3，通过 view 合并最后两维
            layer.weight.data = ops.causal_conv1d_weight_pack(
                layer.weight.view(
                    layer.weight.size(0),
                    layer.weight.size(2), # 潜在风险：若 ndim=1 会越界
                )
            )
        return # 跳过后续 linear 分发

    # 原有 linear 权重分发逻辑不变
```

```
    N, K = layer.weight.size()
    # ...
```

评论区精华

gemini-code-assist[bot] 在 review 中指出了 4 个值得关注的问题：

- 潜在空指针访问：csrc/cpu/sgl-kernels/conv.cpp 中 conv_states->stride(0) 未检查 conv_states 是否有值，可能导致段错误。
- model_config 空指针风险：vllm/platforms/cpu.py 中调用 model_config.get_num_layers_by_block_type 前未确认 model_config 是否 None。
- 张量转置错误：vllm/model_executor/layers/mamba/ops/cpu/gdn_attention.py 中使用 .view() 代替 .transpose() 交换维度，会打乱数据布局。
- 权重维度假设过宽：vllm/model_executor/layers/utils.py 中 ndim != 2 条件过于宽松，可能误处理非 linear 层（如 norm 层）并引发崩溃。 以上问题均未在合并前得到公开回应或修正，但 PR 仍被批准合并，建议后续跟踪修复。
- conv_states 空指针解引用 (correctness): 未公开回复或修改，但 PR 已合并。
- model_config 空值检查缺失 (correctness): 未修改，PR 合并。建议使用者确保 model_config 不为空。
- ssm_state 形状使用 view 错误 (correctness): 未修改，PR 合并。若 v_dim != k_dim 则功能正确性无法保证。
- 权重维度检查 ndim != 2 过于宽泛 (design): 未修改，PR 合并。建议改为更严格的 ndim == 3 检查。

风险与影响

- 风险：
  1. 空指针解引用：conv.cpp 中 conv_states->stride(0) 若 conv_states 为 nullopt 直接崩溃，属于核心路径严重隐患。
  2. model_config 未空检：cpu.py 中若 model_config 为 None 会引发 AttributeError，影响部分初始化场景。
  3. view 代替 transpose：gdn_attention.py 中 ssm_state 形状变换使用 .view() 而非 .transpose()，当 v_dim 与 k_dim 不等时数据错乱，精度将受严重影响。
  4. 权重维度假设：utils.py 中 ndim != 2 检查太宽，若遇到 1D 权重层（如 norm）会触发下标越界，非 AMX 路径也可能回归。
  5. 测试覆盖缺失：无新增测试验证 AMX 内核的正确性与精度，依赖已有集成测试，风险偏高。 - 影响：影响范围限定在 CPU 后端，尤其带有 AMX 指令集的 Intel 服务器。启用 AMX 后会自动禁用 prefix caching 和 chunked prefill，可能降低部分场景的首 Token 延迟但提升计算吞吐。对非 AMX 平台无行为变化。团队需关注 reviewer 指出的潜在缺陷，后续应追加测试和修复。 - 风险标记：潜在空指针解引用，model_config 空值检查缺失，view 替代 transpose 数据错误，权重维度假设可能引起崩溃，缺少测试覆盖，自动禁用 prefix caching 和 chunked prefill

关联脉络

- PR #42311 [Model] [Perf] Use flatten for Qwen3.5's GDN output projection: 本 PR 分支名为 qwen3.5，与 #42311 同属 Qwen3.5 模型 GDN 优化系列，共享线性注意力加速目标。
- PR #42740 [CPU] Specify required KV cache layout for CPU attention backend: 本 PR 调整 conv 状态布局并显式设置 VLLM_SSM_CONV_STATE_LAYOUT=SD，与 #42740 关于 CPU attention 后端 KV cache 布局的规范相关。