PR #27862 完整报告

sgl-project/sglang

Support speculative decoding on CPU

合并时间：2026-07-09 10:27

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27862

执行摘要

- 一句话：支持 CPU 后端四种投机解码算法
- 推荐动作：推荐精读，尤其是后端工厂模式、设备分发策略和 overlap scheduler 的禁用逻辑。后续可分离出 kernel 优化 PR 和模型级别调整 PR。内核向量化可参考 common.h 中的工具函数进行。

功能与动机

CPU 后端此前缺乏投机解码能力，无法充分利用 CPU 推理中常见的并行性。PR 旨在将 GPU 上成熟的投机解码框架引入 CPU，提升 CPU 推理吞吐。见 PR body：'Add initial speculative decoding support to the CPU backend.'

实现拆解

步骤 1：在 sgl-kernel/csrc/cpu/spec.cpp 中实现 CPU 专用投机解码内核，包括 verify_tree_greedy_cpu、build_tree_kernel_efficient_cpu、assign_req_to_token_pool_cpu 等，并在 torch_extension_cpu.cpp 中注册为 PyTorch 算子。 步骤 2：在 sgl-kernel/python/sgl_kernel/speculative.py 中为这些内核添加 Python 封装，同时对 reconstruct_indices_from_tree_mask 添加 is_cpu 分支以调度至 CPU 实现。 步骤 3：扩展 python/sglang/srt/layers/attention/intel_amx_backend.py 中的 IntelAMXAttnBackend，新增 _build_extend_metadata 方法处理 TARGET_VERIFY 模式（从 spec_info 推导 extend_seq_lens 和树掩码），并新增 IntelAMXMultiStepDraftBackend 类管理多步 draft 解码 attention。 步骤 4：修改 python/sglang/srt/speculative/draft_utils.py，在 DraftBackendFactory 的 decode 和 draft-extend 后端注册表中加入 intel_amx 条目，并针对 hybrid_linear_attn 后端在 CPU+AMX 时回落至 intel_amx。 步骤 5：在 server-arg 处理器（speculative_hook.py）中，对 CPU 设备强制禁用 overlap schedule，并对特定组合（如 CPU + topk>1 的 GDN 模型）抛出 ValueError 以防止不支持的配置。 步骤 6：添加单元测试套件：test/registered/cpu/test_spec_kernels.py 覆盖内核正确性（树构建 / 验证 / 缓存管理），test_spec_eagle_cpu.py 和 test_spec_eagle_topk_cpu.py 覆盖 EAGLE 端到端流程，test_spec_cpu_overlap_constraint.py 验证 overlap 禁用逻辑。

关键文件：

- sgl-kernel/python/sgl_kernel/speculative.py（模块 CPU 内核；类别 source；类型 core-logic；符号 verify_tree_greedy_cpu, build_tree_kernel_efficient_cpu,

assign_req_to_token_pool_cpu, build_draft_decode_metadata_cpu）：新增 CPU 内核 Python 封装，是 CPU 投机解码的入口点。

- python/sglang/srt/layers/attention/intel_amx_backend.py（模块 Attention 后端；类别 source；类型 dependency-wiring；符号 _build_extend_metadata, IntelAMXMultiStepDraftBackend, init, init_forward_metadata）：现有 Intel AMX attention 后端扩展支持投机解码元数据生成和多步 draft 解码。
- python/sglang/srt/speculative/draft_utils.py（模块 投机解码框架；类别 source；类型 core-logic；符号 _create_intel_amx_decode_backend, _create_hybrid_linear_attn_decode_backend, _create_hybrid_linear_attn_prefill_backend, _create_intel_amx_prefill_backend）：DraftBackendFactory 中注册 intel_amx 后端，处理 hybrid_linear_attn 回落。
- sgl-kernel/csrc/cpu/spec.cpp（模块 CPU 内核；类别 source；类型 dependency-wiring）：核心 CPU 投机解码内核实现，包含树构建 / 验证、缓存管理、元数据生成等。
- test/registered/cpu/test_spec_kernels.py（模块 测试；类别 test；类型 test-coverage；符号 _topk1_chain_inputs, _gen_draft_tree, _ref_build_tree, _run_build_tree_kernel）：单元测试验证 CPU 内核正确性。
- python/sglang/srt/speculative/eagle_utils.py（模块 投机解码框架；类别 source；类型 core-logic；符号 default_tree_mask_mode）：修改 default_tree_mask_mode 用于 CPU 上默认掩码模式。
- python/sglang/srt/arg_groups/speculative_hook.py（模块 参数钩子；类别 source；类型 configuration；符号 _handle_eagle_family, _handle_ngram）：添加 CPU 设备强制禁用 overlap schedule 逻辑。
- test/registered/unit/spec/test_spec_cpu_overlap_constraint.py（模块 测试；类别 test；类型 test-coverage；符号 _make_spec_args, TestSpecCPUOverlapConstraint, test_cpu_eagle_forces_disable_overlap_schedule, test_cpu_eagle3_forces_disable_overlap_schedule）：验证 CPU 上 overlap 禁用逻辑，确保 CUDA 不受影响。

关键符号：verify_tree_greedy_cpu, build_tree_kernel_efficient_cpu, assign_req_to_token_pool_cpu, build_draft_decode_metadata_cpu, fill_bonus_tokens_cpu, fill_accept_out_cache_loc_cpu, assign_draft_cache_locs_contiguous_cpu, assign_extend_cache_locs_cpu, _build_extend_metadata, IntelAMXMultiStepDraftBackend, _create_intel_amx_decode_backend, _create_intel_amx_prefill_backend, default_tree_mask_mode

关键源码片段

sgl-kernel/python/sgl_kernel/speculative.py

新增 CPU 内核 Python 封装，是 CPU 投机解码的入口点。

```
def verify_tree_greedy_cpu(
    predicts: torch.Tensor, # 输出，形状 [batch, total_tokens]
    accept_index: torch.Tensor, # 输出，位置索引
```

```
    accept_token_num: torch.Tensor, # 输出，接受的 token 数量
    candidates: torch.Tensor, # 输入，候选 token ID
    retrive_index: torch.Tensor, # 输入，检索索引
    retrive_next_token: torch.Tensor, # 输入，下一个 token 映射
    retrive_next_sibling: torch.Tensor, # 输入，兄弟 token 映射
    target_predict: torch.Tensor, # 输入，目标模型预测
) -> None:
    """对 CPU 上的贪婪验证树进行验证。该函数包装了 C++ 内核
    `verify_tree_greedy_cpu`，负责判断哪些候选 token 与目标预测匹配，
    并填充 `predicts` 和 `accept_*` 输出。
    """
    torch.ops.sgl_kernel.verify_tree_greedy_cpu(
        predicts,
        accept_index,
        accept_token_num,
        candidates,
        retrive_index,
        retrive_next_token,
        retrive_next_sibling,
        target_predict,
    )
```

python/sglang/srt/layers/attention/intel_amx_backend.py

现有 Intel AMX attention 后端扩展支持投机解码元数据生成和多步 draft 解码。

```
def _build_extend_metadata(self, forward_batch: ForwardBatch):
    """为当前 forward pass 解析并构建扩展元数据（seq_lens， extend_seq_lens， tree_mask
    等）。
    在 TARGET_VERIFY 模式下，由于 batch 本身不携带 extend_* 字段，
    需要从 spec_info 推导；非投机解码模式则直接传递。
    """
    bs = forward_batch.batch_size
    seq_lens = forward_batch.seq_lens
    tree_mask = None

    if forward_batch.forward_mode.is_target_verify():
        spec_info = forward_batch.spec_info
        if spec_info is None:
            raise RuntimeError(
                "spec_info is unset in TARGET_VERIFY mode; the extend_* "
                "metadata can only be derived from spec_info for "
                "speculative verify batches."
            )
        num_draft_tokens = spec_info.draft_token_num
        # 每个请求的扩展长度均为 num_draft_tokens，构成一个简单 range
        extend_seq_lens = torch.full(
            (bs,), num_draft_tokens, dtype=torch.int32, device=self.device
        )
        extend_start_loc = torch.arange(
```

```
        0,
        bs * num_draft_tokens,
        num_draft_tokens,
        dtype=torch.int32,
        device=self.device,
    )
    seq_lens = forward_batch.seq_lens + num_draft_tokens
    # 树掩码：仅当 tree_topk != 1 时传递显式掩码
    # EAGLE 和 N-Gram 需要树掩码，简单链式跳过
    if spec_info.tree_topk != 1:
        tree_mask = spec_info.custom_mask
    # ... 后续填充 forward_metadata
```

评论区精华

Reviewer Valentine233 和 mingfeima 提出了多项改进建议：内核文件应重命名为 spec.cpp；应使用 AT_DISPATCH_INDEX 支持 int32/int64 输入；CPU 内核应使用向量化加载存储；CPU 调度应放在调用处而非嵌入 triton 实现；contiguous() 调用应避免不必要的内存拷贝；测试时间估计不应过高。作者 htzo 逐一回应并修复了这些点。

- 投机解码内核文件命名 (style): 作者 htzo 采纳，文件已重命名。
- CPU 内核向量化加载 / 存储 (performance): 作者承诺后续 follow-up 优化，当前保留标量实现。
- CPU 内核调度应放在调用处而非 triton 实现中 (design): 作者 htzo 解释了 NPU 也采用了类似模式，但同意未来可重构；当前维持现有方式。
- 抽象 CPU 重叠调度禁用函数 (design): 作者 htzo 采纳，已提取为 disable_overlap_schedule_for_cpu 函数。
- 避免不必要的 contiguous() 内存拷贝 (performance): 作者移除了 layernorm 中的 contiguous()，其他保留但标记为需优化。
- 测试时间设置过高 (testing): 作者 htzo 调整了测试时间。

风险与影响

- 风险：主要风险：(1) 新增 CPU 内核性能未充分向量化，可能成为瓶颈（已标记为 follow-up）；(2) 大量设备条件分支 (if is_cpu()) 可能引入 GPU 侧的回归，需确保 CUDA 路径完全独立；(3) CPU 上强制禁用 overlap scheduler 可能影响其他 CPU 功能；(4) 测试覆盖仅限典型配置，极端情况未覆盖。
- 影响：对 CPU 用户：启用投机解码可显著提升解码吞吐（EAGLE2 实测 1.50x），但仅支持 CPU 上的同步调度路径（overlap 禁用）。对 GPU：无影响，所有更改受 is_cpu() 保护。对团队：PR 体积较大（3500+ 行），增加了维护成本，但设计上保持了与 GPU 框架的对称性，后续易扩展。
- 风险标记：内核性能风险，设备分支覆盖不足，重叠调度强制禁用，测试时间估计不准确

关联脉络

- PR #30265 [AMD] Fix GLM-5.2 MTP Quark excludes: 两者都涉及 MTP 投机解码算法的支持，本 PR 的 MTP 方案可参考其模型调整方式。
- PR #30409 Make CUDA graph disabling PD-role-aware (prefill/decode): 该 PR 涉及调度和 overlap 逻辑，与本 PR 的 overlap 禁用有间接关联。