PR #28527 完整报告

sgl-project/sglang

[Diffusion][CPU] Adding AMX optimizations for CPU platform

合并时间：2026-07-09 10:26

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/28527

执行摘要

- 一句话：为 CPU diffusion 模型引入 AMX 加速优化
- 推荐动作：建议仔细阅读该 PR 的设计思路，特别是 AMXAttentionBackend 的轻量实现和 linear.py 中的条件分支设计。但由于该 PR 已回滚，直接应用存在风险。可跟踪后续修复 PR（如 #30717 和可能的重新合入版本），待验证确无回归后再考虑采纳。重点关注 FSDP 加载时的权重处理逻辑，避免重复调用。

功能与动机

基于此前在 LLM 模型中已验证的 AMX 优化方案（见 #20816），将同类加速技术扩展到 diffusion 模型，以显著提升 CPU 平台的推理性能。PR body 中明确指出：'This pr takes parts of follow-ups (mentioned in https://github.com/sgl-project/sglang/pull/20816) to bring key AMX based optimizations (scoping from LLM models) for CPU platforms'。

实现拆解

1. 创建 AMX Attention 后端（amx_attn.py）：实现 AMXAttentionBackend 和 AMXATTNImpl，分别继承 AttentionBackend/AttentionImpl，在 forward 中调用 torch.ops.sgl_kernel.flash_attn_varlen_func 进行加速，可支持的头大小集合通过 get_supported_head_sizes 限定。
2. 线性层 AMX 支持（linear.py）：在 UnquantizedLinearMethod 中新增 process_weights_after_loading 方法，当 CPU 支持 AMX 时调用 _amx_process_weight_after_loading 对权重进行 VNNI 格式打包；在 apply 方法中，当 use_intel_amx_backend(layer) 为真时调用 torch.ops.sgl_kernel.weight_packed_linear 算子。
3. 平台选择逻辑（cpu.py）：在 CpuPlatform.get_attn_backend_cls_str 中增加 AttentionBackendEnum.AMX_ATTN 的识别，当 CPU 具备 AMX 能力时优先返回 AMXAttentionBackend 路径，否则回退到 SDPABackend。
4. FSDP 加载流程加固（fsdp_load.py、text_encoder_loader.py）：在权重后处理阶段增加对 process_weights_after_loading 的二次调用，并包裹在 device_loading_context 中以确保参数在目标设备上完成打包，兼容 CPU offload 场景。
5. C++ 内核适配（flash_attn.cpp）：flash_attn_varlen_func 的 C++ 接口新增可选参数 sm_scale，允许外部传入 softmax 缩放系数，默认仍为 1/sqrt(head_size)。

6. 测试增强（test_flash_attn.py）：添加 sm_scale 为 None 和给定浮点值时的测试用例，确保默认行为与显式传值一致。
7. 辅助配置（interface.py、vision.py、wanvae.py 等）：枚举类增加 AMX_ATTN；视觉 attention 中启用 AMX；VAE 模型启用 channel last 3d 布局。

关键文件：

- python/sglang/multimodal_gen/runtime/layers/attention/backends/amx_attn.py（模块 注意力后端；类别 source；类型 core-logic；符号 AMXAttentionBackend, get_supported_head_sizes, get_enum, get_impl_cls）：新增文件，核心 AMX 注意力后端实现，定义了 AMXAttentionBackend 和 AMXATTNImpl。
- python/sglang/multimodal_gen/runtime/layers/linear.py（模块 线性层；类别 source；类型 core-logic；符号 process_weights_after_loading）：修改文件，在 UnquantizedLinearMethod 中添加 AMX 权重打包和 AMX 线性计算分支。
- python/sglang/multimodal_gen/runtime/platforms/cpu.py（模块 平台适配；类别 source；类型 dependency-wiring）：修改平台选择逻辑，当 CPU 支持 AMX 时优先返回 AMXAttentionBackend。
- python/sglang/multimodal_gen/runtime/loader/fsdp_load.py（模块 加载器；类别 source；类型 dependency-wiring）：修改文件，增加二次 process_weights_after_loading 调用但导致重复，后在 #30717 修复。
- python/sglang/multimodal_gen/runtime/loader/component_loaders/text_encoder_loader.py（模块 加载器；类别 source；类型 dependency-wiring）：修改文件，类似 fsdp_load.py，增加 process_weights_after_loading 二次调用。
- sgl-kernel/csrc/cpu/flash_attn.cpp（模块 内核；类别 source；类型 core-logic）：修改 C++ 内核，flash_attn_varlen_func 增加可选 sm_scale 参数。

关键符号：AMXAttentionBackend, AMXATTNImpl, forward, process_weights_after_loading, apply, get_attn_backend_cls_str, flash_attn_varlen_func

关键源码片段

python/sglang/multimodal_gen/runtime/layers/attention/backends/amx_attn.py

新增文件，核心 AMX 注意力后端实现，定义了 AMXAttentionBackend 和 AMXATTNImpl。

```
# SPDX-License-Identifier: Apache-2.0
# AMX Attention backend for CPU diffusion models

import torch
from sglang.multimodal_gen.runtime.layers.attention.backends.attention_backend import (
    AttentionBackend,
    AttentionImpl,
    AttentionMetadata,
)
from sglang.multimodal_gen.runtime.platforms import AttentionBackendEnum
from sglang.multimodal_gen.runtime.utils.logging_utils import init_logger
```

```
logger = init_logger(__name__)
# 使用 AMX 优化的 flash attention 变长函数
flash_attn_varlen_func = torch.ops.sgl_kernel.flash_attn_varlen_func


class AMXAttentionBackend(AttentionBackend):
    """AMX attention backend，仅支持特定头大小"""
    accept_output_buffer: bool = True

    @staticmethod
    def get_supported_head_sizes() -> list[int]:
        # AMX 加速支持的头大小列表，步长为 32
        return [32, 64, 96, 128, 160, 192, 224, 256]

    @staticmethod
    def get_enum() -> AttentionBackendEnum:
        return AttentionBackendEnum.AMX_ATTN

    @staticmethod
    def get_impl_cls() -> type["AMXATTNImpl"]:
        return AMXATTNImpl


class AMXATTNImpl(AttentionImpl):
    """AMX attention 实现，调用 flash_attn_varlen_func"""

    def __init__(
        self,
        num_heads: int,
        head_size: int,
        causal: bool,
        softmax_scale: float,
        num_kv_heads: int | None = None,
        prefix: str = "",
        **extra_impl_args,
    ) -> None:
        self.causal = causal
        self.softmax_scale = softmax_scale

    def forward(
        self,
        query: torch.Tensor,
        key: torch.Tensor,
        value: torch.Tensor,
        attn_metadata: AttentionMetadata,
    ) -> torch.Tensor:
        # 假设输入形状为 (1, seq_len, num_heads, head_size)
        max_seqlen_q = query.shape[1]
        max_seqlen_k = key.shape[1]
```

```
        # 调用 AMX 优化的变长 flash attention
        return flash_attn_varlen_func(
            query[0],
            key[0],
            value[0],
            torch.tensor([0, max_seqlen_q]).to(torch.int),
            torch.tensor([0, max_seqlen_k]).to(torch.int),
            max_seqlen_q,
            max_seqlen_k,
            self.causal,
            self.softmax_scale,
        ).unsqueeze(0)
```

python/sglang/multimodal_gen/runtime/layers/linear.py

修改文件，在 UnquantizedLinearMethod 中添加 AMX 权重打包和 AMX 线性计算分支。

```
# 线性层中 AMX 加速的支持
from sglang.srt.layers.amx_utils import _amx_process_weight_after_loading
from sglang.srt.utils import (
    cpu_has_amx_support,
    is_cpu,
    use_intel_amx_backend,
)

_is_cpu_amx_available = cpu_has_amx_support()
_is_cpu = is_cpu()

class UnquantizedLinearMethod(LinearMethodBase):
    """无量化线性方法，新增AMX支持"""

    def create_weights(self, layer, input_size_per_partition, output_partition_sizes, input_size,
    output_size, params_dtype, **extra_weight_attrs):
        weight = Parameter(
            torch.empty(sum(output_partition_sizes), input_size_per_partition, dtype=params_dtype)
            ,
            requires_grad=False,
        )
        set_weight_attrs(weight, {"input_dim": 1, "output_dim": 0})
        layer.register_parameter("weight", weight)
        set_weight_attrs(weight, extra_weight_attrs)

    def process_weights_after_loading(self, layer: torch.nn.Module) -> None:
        # 如果 CPU 支持 AMX，则对权重进行 VNNI 格式打包
        if _is_cpu and _is_cpu_amx_available:
            _amx_process_weight_after_loading(layer, ["weight"])

    def apply(self, layer, x, bias=None):
        # 如果启用 AMX 后端，使用 weight_packed_linear
        if use_intel_amx_backend(layer):
```

```
            x_shapes = x.shape
            if len(x_shapes) == 3:
                x = x.view(-1, x.shape[-1])
            output = torch.ops.sgl_kernel.weight_packed_linear(
                x.to(layer.weight.dtype),
                layer.weight,
                bias,
                True, # is_vnni 标记，表明权重已打包
            )
            if len(x_shapes) == 3:
                output = output.view(x_shapes[0], x_shapes[1], -1)
            return output
        # 否则使用标准线性层
        output = (
            F.linear(x, layer.weight, bias)
            if IS_AMP_SUPPORTED or bias is None
            else F.linear(x, layer.weight, bias.to(x.dtype))
        )
        return output
```

python/sglang/multimodal_gen/runtime/platforms/cpu.py

修改平台选择逻辑，当 CPU 支持 AMX 时优先返回 AMXAttentionBackend。

```
# CPU 平台中 attention 后端选择逻辑
from sglang.srt.utils import cpu_has_amx_support, is_cpu

_is_cpu_amx_available = cpu_has_amx_support()
_is_cpu = is_cpu()

class CpuPlatform(Platform):
    # ... 其他方法 ...

    @classmethod
    def get_attn_backend_cls_str(
        cls,
        selected_backend: AttentionBackendEnum | None,
        head_size: int,
        dtype: torch.dtype,
    ) -> str:
        # 如果用户选择非 SDPA/AMX 后端，发出警告并自动选择
        if selected_backend not in (
            None,
            AttentionBackendEnum.TORCH_SDPA,
            AttentionBackendEnum.AMX_ATTN,
        ):
            logger.warning(
                "%s is not supported on CPU; falling back to auto selection SDPA or AMX_ATTN",
                selected_backend,
            )
```

```
    # CPU 且支持 AMX 时优先使用 AMX 后端
    if _is_cpu and _is_cpu_amx_available:
        logger.info("Using AMX Attention backend for CPU.")
        return "sglang.multimodal_gen.runtime.layers.attention.backends.amx_attn.
        AMXAttentionBackend"
    logger.info("Using Torch SDPA backend for CPU.")
    return "sglang.multimodal_gen.runtime.layers.attention.backends.sdpa.SDPABackend"
```

评论区精华

1. 测试覆盖要求（mingfeima）：「add test case when sm_scale is None and a given value. some cases are not guarded in the test cases.」作者 jianan-gu 已按要求添加测试。
2. 避免重复调用（mickqian）：「we have an existing quant_method.process_weights_after_loading call in L321-L330, please avoid duplicating it」作者承认问题并提交修复 PR #30717。
3. CI 断裂导致回滚（mickqian 在 issue 评论中）：「this breaks CI, reverting in #30716」表明该 PR 上线后引入 CI 失败，最终被回滚。

- Add sm_scale test cases (testing): jianan-gu: sure, have added.
- Avoid duplicate process_weights_after_loading in fsdp_load.py (correctness): jianan-gu: Thanks for pointing that, have submitted changes to avoid such duplicating https://github.com/sgl-project/sglang/pull/30717

风险与影响

- 风险：
  1. CI 稳定性风险：该 PR 直接导致 CI 失败，最终被 #30716 回滚，说明 AMX 优化在部分测试环境或配置下存在未预料的兼容性问题（可能涉及 FSDP 加载或特定模型）。
  2. 重复权重处理风险：mickqian 指出的 fsdp_load.py 中 process_weights_after_loading 重复调用虽已在 #30717 修复，但在初始版本中可能导致权重打包两次，引发显存错误或计算结果异常。
  3. AMX 后端覆盖不足：新增的 AMXAttentionBackend 仅在特定头大小（32-256，步长 32）下可用，若后续模型使用不在支持范围内的头大小则会静默回退到 SDPA，可能造成用户预期之外的性能差异。
  4. Channel last 3d 变更：wanvae.py 中启用 channel last 可能改变张量内存布局，若与其他算子（如卷积）的期望布局不符，可能导致额外重排开销或崩溃。 - 影响：影响范围：限于 CPU 平台的 diffusion 推理用户。性能影响巨大（最高 10 倍加速），但稳定性风险也高（已被回滚）。团队需要在修复后续问题后重新评估并合入。该 PR 展示了 CPU AMX 优化在 diffusion 模型上的巨大潜力，但当前版本尚不宜在生产环境使用。 - 风险标记：CI 断裂已回滚，FSDP 权重处理重复，AMX 后端条件覆盖有限

关联脉络

- PR #20816 AMX optimizations for LLM models: 该 PR 是此 PR 的起点和基础，表明 AMX 优化从 LLM 扩展到 diffusion。
- PR #30716 [Diffusion] Revert CPU AMX optimizations: 回滚此 PR，因为 CI 被破坏。
- PR #30717 Fix duplicate process_weights_after_loading in fsdp_load: 修复此 PR 中 fsdp_load.py 的重复权重处理问题，是此 PR 的后续修复。