

PR #33452 完整报告

sgl-project/sglang

[CPU] [Diffusion] Add fused scale-shift and norm kernels for CPU

合并时间: 2026-09-16 10:01

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/33452>

执行摘要

- 一句话: 为 CPU 扩散模型新增三个融合 norm / scale-shift 内核
- 推荐动作: 值得精读, 尤其是在非 CUDA 平台写融合内核的工程师。重点关注四个设计决策: 一是用 DiffusionNormMode 编译期 tag + DiffusionNormTraits 让 RMSNorm/LayerNorm 共享一套 row kernel, 避免复制两遍几乎相同的向量循环; 二是把广播语义全部推给 Python 的 expand_scale_shift_cpu_param, 让 C++ 只面对 stride_c $\in \{0, 1\}$, 这是把「组合爆炸」挡在语言边界之外的好例子; 三是用「两个 gate 指针」而不是 gate_t 模板派发来兼容 Hunyuan (activation dtype) 与 Wan (FP32) 两种 gate, 代价是函数多一个参数、收益是少一层 dispatch; 四是刻意复刻 CUDA 的 activation-dtype 边界以保证跨设备数值一致, 读代码时要留意那两处 convert_from_float_ext / static_cast<scalar_t> 往返并非冗余。读法与风险相关: normd.cpp 前半 (dispatch 与 helper) 与 layernorm.py 的两处 forward_cpu 是主干, torch_extension_cpu.cpp 只是样板注册可略读; 建议同时留意 custom_op.py 中 forward_cpu 默认回退的语义翻转, 以及测试未覆盖的非连续输入与 4D 广播分支。

功能与动机

PR body 明确说明动机: 「adds three fused CPU kernels for diffusion workloads」并解释目的是「fuse scale/shift, normalization, and residual operations to reduce intermediate tensor overhead and improve CPU inference performance」, 同时补齐单元测试。作者给出的基线对比是 E2E 1.16x、DenoisingStage 1.20x (Wan2.1-T2V-1.3B、TP=4、480x832x33 帧、15 步)。除此之外, reviewer mingfeima 的评审把「减少中间张量、提升 CPU 推理性能」进一步转化为工程约束: 不要把 modulate 折进通用 NormReduceGeneric、不要做 4 路 dtype 笛卡尔积、不要在 CPU 侧强制 contiguous, 从而让内核既快又易读。

实现拆解

1. 新建 diffusion 专用内核文件 `python/sglang/kernels/aot/csrc/cpu/normd.cpp` (+662)。用编译期 tag `enum class DiffusionNormMode { RMSNorm, LayerNorm }` 与 `DiffusionNormTraits<M> (has_mean / has_bias)` 把 rms 与 layer 两条路径收进同一套 row kernel, 配合 `if constexpr` 消除运行期分支; `DISPATCH_DIFFUSION_NORM_TYPE(norm_type, name, ...)` 宏把字符串 "rms"/"layer" 收敛为唯一入口。三个算符共用 `parallel_for_rows(B, S, D, row_fn)` 的并行方案, 以及 `ModulationParam<T>` (数据指针 + 三段 stride)、`load_param_vec2` (stride_c 为 0 时退化为标量广播)、

`apply_scale_shift_vec`、`apply_residual_gate_vec`、`apply_norm_modulate_row` 等 helper。

这样改的直接原因是 reviewer 要求「do not fold modulate into the existing RMS/LayerNorm kernel. Keep NormReduceGeneric as it was」，以及「norm.cpp is getting bigger and bigger. You can move to another file, name it after normd.cpp」。

影响面是整个 CPU AOT 内核目录新增 diffusion 语义，与 SRT 侧通用 norm 内核解耦。

2. 注册三个 CPU 算子 `python/sglang/kernels/aot/csrc/cpu/torch_extension_cpu.cpp` (+54)
。声明并 `m.impl` 注册 `fused_scale_shift_cpu`、`fused_norm_scale_shift_cpu`、`fused_scale_residual_norm_scale_shift_cpu`；后两者的 `weight / bias / gate` 均为 `Tensor?` 可选，`norm_type` 以 `str` 传入、`eps` 为 `float`，融合算符返回 `(Tensor, Tensor)`（输出 + `residual_out`）。这是 Python 侧 `torch.ops.sgl_kernel.*` 能被解析的前提。
3. Python 侧形状归一与平台派发 `python/sglang/kernels/ops/diffusion/modulate/scale_shift_triton.py` (+61/-1)。新增 `expand_scale_shift_cpu_param(tensor, x)` 把 0D/1D/2D/3D/4D 调制张量统一成 `[B, L, C]`（其中 4D `[B, F, 1, C]` 会按帧展开），并对非法形状抛 `ValueError`；新增 `_fuse_scale_shift_kernel_cpu`，把算子平台表的 `cpu=` 从 `lazy_fallback("torch", "fuse_scale_shift_kernel_native")` 换成本函数。这样 C++ 只需处理 `stride_c == 0` 或 `1`，正是 reviewer 要求的「Prefer Python expand over the 1D-4D ScaleShiftBroadcastParams switch」。
4. 运行时接入融合路径 `python/sglang/multimodal_gen/runtime/layers/layernorm.py` (+61)
。为 `_NormScaleShift` 与 `_ScaleResidualNormScaleShift` 各加一个 `forward_cpu`：从 `self.norm` 取 `weight / bias`，把 `gate == 1` 映射为 `None`（cross-attn 场景退化成纯 residual add）、其它非张量 `gate` 回落 `forward_native`，再调 `torch.ops.sgl_kernel.*`；非连续输入不再强制 `.contiguous()`。配套地 `python/sglang/multimodal_gen/runtime/layers/custom_op.py` (+5/-2) 把 `forward_cpu` 默认实现由 `forward_cuda` 改为 `forward_native`，并在 `dispatch_forward` 里补 `elif current_platform.is_cpu(): return self.forward_cpu`，使各算符可以单独覆盖 CPU 实现而不影响其它平台。
5. 测试与 CI 配套 `test/registered/unit/cpu/test_diffusion_norm.py` (+184)。
`DTTYPE_PAIRS` 覆盖 BF16/FP16 激活 × BF16/FP32 参数的四种组合，`broadcast_c` 参数显式断言 `stride(2) == 0` 与 `== 1` 两条访存路径；`test_fused_scale_residual_norm_scale_shift` 用 `("input", None, "input", "rms")`、`("fp32", torch.float32, "input", "layer")`、`(None, None, "fp32", "layer")` 三组参数对应 Hunyuan / Wan self-attn / Wan cross-attn 的真实调用形态，参考实现刻意复刻「norm FP32 → activation dtype → scale/shift」的 CUDA 边界。测试分别 `register_cpu_ci(est_time=5, suite="stage-a-test-cpu-intel")` 与 `(est_time=10, suite="base-b-test-cpu-arm64")`，覆盖 Intel 与 ARM64 两条 CPU 流水线。

关键文件：

- `python/sglang/kernels/aot/csrc/cpu/normd.cpp`（模块 融合内核；类别 `source`；类型 `core-logic`；符号 `DiffusionNormMode`, `DiffusionNormTraits`, `DISPATCH_DIFFUSION_NORM_TYPE`, `ModulationParam`）：本 PR 的核心：从 `norm.cpp` 拆出的独立文件，用编译期模式 `tag + traits` 让 `RMSNorm/LayerNorm` 共享 `row kernel`，实现三个融合算符的全部 CPU 计算逻辑，是性能收益与设计讨论的落点。
- `test/registered/unit/cpu/test_diffusion_norm.py`（模块 内核测试；类别 `test`；类型 `test-coverage`；符号 `TestDiffusionNorm`, `rmsnorm_ref`, `test_fused_scale_shift`,

test_fused_scale_residual_norm_scale_shift)：新增回归测试：用 DTYPE_PAIRS 覆盖 BF16/FP16 激活 × BF16/FP32 参数，用 broadcast_c 显式验证 stride 0/1 两条访存路径，并用三组参数对应 Hunyuan / Wan self-attn / Wan cross-attn 的真实调用形态。

- python/sglang/multimodal_gen/runtime/layers/layernorm.py（模块 归一化层；类别 source；类型 core-logic；符号 forward_cpu）：运行时接入点：为 _NormScaleShift 与 _ScaleResidualNormScaleShift 新增 forward_cpu，决定何时走融合算子、何时回落 forward_native，并在此完成 gate/scale/shift 的形状归一。
- python/sglang/kernels/ops/diffusion/modulate/scale_shift_triton.py（模块 形状广播；类别 infra；类型 infrastructure；符号 expand_scale_shift_cpu_param, _fuse_scale_shift_kernel_cpu）：Python 侧形状契约与平台派发：expand_scale_shift_cpu_param 把 0D–4D 广播统一为 [B, L, C]，并把 fuse_scale_shift_kernel 的 cpu 分支从 native 回落切换到新内核。
- python/sglang/kernels/aot/csrc/cpu/torch_extension_cpu.cpp（模块 算子注册；类别 source；类型 api-registration；符号 fused_scale_shift_cpu, fused_norm_scale_shift_cpu, fused_scale_residual_norm_scale_shift_cpu）：算子注册入口：为三个融合算符补函数声明与 m.def/m.impl，是把 C++ 实现暴露给 Python torch.ops.sgl_kernel 的必要样板，也是 review 中「CPU_DISPATCH_FLOATING_TYPES 冗余」讨论的同一文件族。
- python/sglang/multimodal_gen/runtime/layers/custom_op.py（模块 平台派发；类别 source；类型 platform-dispatch；符号 forward_cpu, dispatch_forward）：框架级派发变更：forward_cpu 默认实现由 forward_cuda 改为 forward_native，并在 dispatch_forward 中补上 is_cpu 分支，使上层算符能按平台覆盖 CPU 实现；影响 multimodal_gen 全部 CPU 算符。

关键符号：fused_scale_shift_cpu, fused_norm_scale_shift_cpu, fused_scale_residual_norm_scale_shift_cpu, fused_scale_shift_row, fused_norm_scale_shift_row, apply_norm_modulate_row, apply_scale_shift_vec, apply_residual_gate_vec, load_param_vec2, parallel_for_rows, DISPATCH_DIFFUSION_NORM_TYPE, expand_scale_shift_cpu_param, _fuse_scale_shift_kernel_cpu, _NormScaleShift.forward_cpu, _ScaleResidualNormScaleShift.forward_cpu, CustomOp.dispatch_forward

关键源码片段

test/registered/unit/cpu/test_diffusion_norm.py

新增回归测试：用 DTYPE_PAIRS 覆盖 BF16/FP16 激活 × BF16/FP32 参数，用 broadcast_c 显式验证 stride 0/1 两条访存路径，并用三组参数对应 Hunyuan / Wan self-attn / Wan cross-attn 的真实调用形态。

```
# test/registered/unit/cpu/test_diffusion_norm.py
# 同时挂到 Intel x86 与 ARM64 两条 CPU 流水线，保证跨架构可用性。
register_cpu_ci(est_time=5, suite="stage-a-test-cpu-intel")
register_cpu_ci(est_time=10, suite="base-b-test-cpu-arm64")
```

激活 dtype 与调制参数 dtype 的四种组合，覆盖 BF16/FP16 激活与 BF16/FP32 参数；

按 reviewer 建议提到 parametrize 之上，避免在每个测试里重复内联列表。

```
DTYPE_PAIRS = [
    (torch.bfloat16, torch.bfloat16),
    (torch.bfloat16, torch.float32),
    (torch.float16, torch.float16),
    (torch.float16, torch.float32),
]
```

```
class TestDiffusionNorm:
```

```
    @pytest.mark.parametrize("input_dtype,param_dtype", DTYPE_PAIRS)
```

```
    @pytest.mark.parametrize("broadcast_c", [False, True])
```

```
    def test_fused_scale_shift(self, input_dtype, param_dtype, broadcast_c):
```

```
        B, S, D = 2, 4, 67 # D 取 67: 故意不是向量宽度整数倍，以走到标量尾循环
```

```
        x = torch.randn(B, S, D, dtype=input_dtype)
```

```
        if broadcast_c:
```

```
            # hidden 维广播 -> kernel 侧 stride_c == 0 的标量分支
```

```
            scale = torch.randn(B, 1, 1, dtype=param_dtype)
```

```
            shift = torch.randn(B, S, 1, dtype=param_dtype)
```

```
        else:
```

```
            # 正常向量加载 -> kernel 侧 stride_c == 1
```

```
            scale = torch.randn(B, 1, D, dtype=param_dtype)
```

```
            shift = torch.randn(B, S, D, dtype=param_dtype)
```

```
        # 显式断言两条路径真的被展开成 stride 0 / 1，避免测试假绿
```

```
        scale_expanded = scale.expand_as(x)
```

```
        shift_expanded = shift.expand_as(x)
```

```
        assert scale_expanded.stride(2) == (0 if broadcast_c else 1)
```

```
        out = torch.ops.sgl_kernel.fused_scale_shift_cpu(x, scale_expanded, shift_expanded, 1.0)
```

```
        # 参考实现全程 fp32 计算后再降回激活 dtype，与 kernel 的 scale_constant 语义一致
```

```
        ref = (x.float() * (1.0 + scale.float()) + shift.float()).to(input_dtype)
```

```
        torch.testing.assert_close(
```

```
            out, ref, atol=precision[input_dtype], rtol=precision[input_dtype]
```

```
        )
```

```
    @pytest.mark.parametrize("input_dtype", [torch.bfloat16, torch.float16])
```

```
    @pytest.mark.parametrize(
```

```
        "gate_type,norm_dtype,param_type,norm_type",
```

```
        [
```

```
            ("input", None, "input", "rms"), # Hunyuan / Hunyuan3D: gate 为激活 dtype
```

```
            ("fp32", torch.float32, "input", "layer"), # Wan / LingBot self-attn: gate 为 FP32
```

```
            (None, None, "fp32", "layer"), # Wan / LingBot cross-attn: 无 gate
```

```
        ],
```

```
    )
```

```
    def test_fused_scale_residual_norm_scale_shift(
```

```
        self, input_dtype, gate_type, norm_dtype, param_type, norm_type
```

```

):
# ... 构造 x / residual / gate / weight / bias 后调用:
out, residual_out = torch.ops.sgl_kernel.fused_scale_residual_norm_scale_shift_cpu(
    residual, x, gate_expanded, weight, bias,
    scale_expanded, shift_expanded, norm_type, eps=eps,
)

# 参考值同样先算 residual, 再在 fp32 里归一化,
# 最后刻意做 normalized.to(input_dtype).float() 往返以复刻 CUDA 的 activation 边界。
# 注意断言的是两个输出: residual_out 与 modulate 之后的 out。

```

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

运行时接入点: 为 `_NormScaleShift` 与 `_ScaleResidualNormScaleShift` 新增 `forward_cpu`, 决定何时走融合算子、何时回落 `forward_native`, 并在此完成 `gate/scale/shift` 的形状归一。

```

# python/sglang/multimodal_gen/runtime/layers/layernorm.py
# 顶层导入新增的广播辅助函数, 保证 C++ 侧只看 stride_c == 0 / 1。
from sglang.kernels.ops.diffusion.modulate.scale_shift_triton import (
    expand_scale_shift_cpu_param,
)

```

```

class _ScaleResidualNormScaleShift(CustomOp):
    def forward_cpu(
        self,
        residual: torch.Tensor,
        x: torch.Tensor,
        gate: torch.Tensor | int,
        shift: torch.Tensor,
        scale: torch.Tensor,
    ) -> tuple[torch.Tensor, torch.Tensor]:
        # norm 的可学习参数挂在 self.norm 上; rms 路径没有 weight/bias,
        # getattr 拿不到就传 None, kernel 侧用 nullptr 分支跳过。
        weight = getattr(self.norm, "weight", None)
        bias = getattr(self.norm, "bias", None)

        if isinstance(gate, torch.Tensor):
            gate_tensor = gate
        elif gate == 1:
            gate_tensor = None # cross-attention 的 gate 恒为 1, 退化成纯 residual add
        else:
            # 其它非常规 gate 形态不冒险, 直接回落 PyTorch native 实现
            return self.forward_native(residual, x, gate, shift, scale)

        # 形状归一在这里一次做完: scalar/1D/2D/3D/4D -> [B, L, C],
        # 于是 C++ 只需要处理 stride_c 为 0 或 1 两种情况。
        scale = expand_scale_shift_cpu_param(scale, x)
        shift = expand_scale_shift_cpu_param(shift, x)
        if gate_tensor is not None:

```

```

        gate_tensor = expand_scale_shift_cpu_param(gate_tensor, x)

# residual / x 不调用 .contiguous(): kernel 只要求最后一维连续,
# 强行 contiguous 会在非连续输入时引入隐式 memcpy (reviewer 指出的性能退化点)。
# weight / bias 则必须连续才能走向量加载, 用 _ensure_contiguous 兜住。
return torch.ops.sgl_kernel.fused_scale_residual_norm_scale_shift_cpu(
    residual,
    x,
    gate_tensor,
    _ensure_contiguous(weight),
    _ensure_contiguous(bias),
    scale,
    shift,
    self.norm_type,
    self.eps,
)

class _NormScaleShift(CustomOp):
    def forward_cpu(
        self,
        x: torch.Tensor,
        shift: torch.Tensor,
        scale: torch.Tensor,
    ) -> torch.Tensor:
        # 无 residual、无 gate 的简化形态: norm + scale-shift 一次融合,
        # 对应 Wan / LingBot / Helios 的 LayerNormScaleShift 路径。
        weight = getattr(self.norm, "weight", None)
        bias = getattr(self.norm, "bias", None)

        scale = expand_scale_shift_cpu_param(scale, x)
        shift = expand_scale_shift_cpu_param(shift, x)

        return torch.ops.sgl_kernel.fused_norm_scale_shift_cpu(
            x,
            _ensure_contiguous(weight),
            _ensure_contiguous(bias),
            scale,
            shift,
            self.norm_type, # "rms" / "layer", 由 C++ 派发宏消费
            self.eps,
        )

```

[python/sglang/kernels/ops/diffusion/modulate/scale_shift_triton.py](#)

Python 侧形状契约与平台派发: `expand_scale_shift_cpu_param` 把 0D-4D 广播统一为 [B, L, C], 并把 `fuse_scale_shift_kernel` 的 cpu 分支从 native 回落切换到新内核。

[python/sglang/kernels/ops/diffusion/modulate/scale_shift_triton.py](#)

```

def expand_scale_shift_cpu_param(tensor: torch.Tensor, x: torch.Tensor) -> torch.Tensor:
    """把调制张量归一到 [B, L, C], 让 C++ 内核只需处理 stride_c == 0 / 1。"""
    B, L, C = x.shape

    if tensor.numel() == 1:
        # 标量 -> 完全广播, 最终 stride_c == 0
        return tensor.reshape(1, 1, 1).expand(B, L, C)

    if tensor.dim() == 1:
        if tensor.shape[0] != C:
            raise ValueError(f"1D modulation tensor must have shape [{C}]")
        tensor = tensor.reshape(1, 1, C)
    elif tensor.dim() == 2:
        tensor = tensor[:, None, :]
    elif tensor.dim() == 3:
        pass
    elif tensor.dim() == 4:
        # Wan 系列按帧取调制参数: [B, F, 1, C] -> [B, L, C]
        if tensor.shape[2] != 1:
            raise ValueError("4D modulation tensor must have shape [B, F, 1, C]")
        num_frames = tensor.shape[1]
        if L % num_frames != 0:
            raise ValueError("sequence length must be divisible by num_frames")
        frame_seqlen = L // num_frames
        tensor = tensor.expand(
            tensor.shape[0], num_frames, frame_seqlen, tensor.shape[-1]
        ).reshape(tensor.shape[0], L, tensor.shape[-1])
    else:
        raise ValueError("modulation tensor must be scalar or 1D/2D/3D/4D")

    # expand 而不是 repeat: 不复制数据, kernel 侧看到 stride_c == 0 或 1
    return tensor.expand(B, L, C)


def _fuse_scale_shift_kernel_cpu(
    x: torch.Tensor,
    scale: torch.Tensor,
    shift: torch.Tensor,
    scale_constant: float = 1.0,
    block_l: int = 128,
    block_c: int = 128,
) -> torch.Tensor:
    # block_l / block_c 是 Triton 的网格参数, 对 CPU 内核无意义, 显式丢弃
    import sgl_kernel # noqa: F401

    del block_l, block_c

    scale = expand_scale_shift_cpu_param(scale, x)
    shift = expand_scale_shift_cpu_param(shift, x)

```

```
return torch.ops.sgl_kernel.fused_scale_shift_cpu(x, scale, shift, scale_constant)
```

```
# 平台派发表: CPU 从 lazy_fallback("torch", "fuse_scale_shift_kernel_native")
# 切换到真实融合内核 (其余平台保持原样)。
# cpu=_fuse_scale_shift_kernel_cpu,
```

评论区精华

评审由 maintainer mingfeima 主导, 前后两轮 CHANGES_REQUESTED, 共 8 条 review comment, 最终 APPROVED。核心交锋集中在三点:

1) 架构分层之争。mingfeima 第一轮直接要求: 「[simplify] python/sglang/kernels/aot/csrc/cpu/norm.cpp (NormReduceGeneric): do not fold modulate into the existing RMS/LayerNorm kernel. has_scale_shift + param_t/gate_t/norm_t plus apply_scale_shift(..., scale_constant=0) for residual_gate makes the shared path much harder to read. Keep NormReduceGeneric as it was; implement the three fused ops as their own row kernels that call apply_scale_shift_*」, 并同时要求「Drop CPU_DISPATCH_FLOATING_TYPES in common.h」 「Remove the const_cast on residual now that residual_output exists」。作者接受了「独立 row kernel」的方向, 最终以新文件 normd.cpp 落地。

2) dtype 组合爆炸。mingfeima 评论: 「overall this patch is too complexed. you can list all the feasible combinations of dtypes for all the inputs in a table. And then simplify the dtype dispatch accordingly.」作者用一张 4 行表格回应了 HunyuanVideo/Hunyuan3D、Wan/LingBot/Helios、Wan self-attn、Wan cross-attn 四种真实路径的 (input/residual, scale/shift, gate, norm weight/bias) 组合, 并说明「I have removed the unnecessary dtype dispatch. Activations are BF16/FP16, scale/shift use the input dtype or FP32, and norm weight/bias are FP32 or None. Gate needs both input-dtype support for Hunyuan and FP32 support for Wan, so I use two gate pointers (const scalar_tand const float) instead of gate_t dispatch.」——这条「双 gate 指针替代模板派发」是整份讨论里最有技术含量的一处折中。

3) 性能与诚实边界。mingfeima 对 layernorm.py 两处 forward_cpu 都提了同一条: 「the kernel checks lastdim contiguous, but here x is forced to be contiguous() which may introduce implicit memcpy which is performance downgrade.」, 作者回复「Done!Thanks!」, 最终版本 residual/x 直接传入 (仅 weight/bias 走 _ensure_contiguous)。

此外 mingfeima 主动给了可落地的重构代码: DISPATCH_DIFFUSION_NORM_TYPE 宏、parallel_for_rows 仿函数, 以及抽取 load_param_vec2 / apply_residual_gate_vec / apply_norm_modulate_row 三个 helper 的完整建议; 测试侧则要求把 DTYPE_PAIRS 提到 parametrize 之上。所有评论线程均以作者「Done!」收尾, 无遗留未决项。

- 不要把 modulate 折进通用 NormReduceGeneric, 三个融合算符独立成 row kernel (design): 作者按此重构: 通用 norm 内核保持原样, 三个融合算符各自实现 row kernel 并共用 helper; 最终以独立文件 normd.cpp 落地, const_cast 因 residual_output 作为返回值而消失。

- dtype 组合爆炸与派发简化（列出可行组合表 + 双 gate 指针）（design）：采纳：去掉 4 路笛卡尔积与 gate_t/norm_t 模板派发，改为由真实模型需求倒推的有限组合 + 两个 gate 指针。
- norm.cpp 体积膨胀，拆出独立文件（design）：新增 python/sglang/kernels/aot/csrc/cpu/normd.cpp（662 行）承载 diffusion 专用融合算符，通用 norm 逻辑留在原文件。
- 抽取 DISPATCH_DIFFUSION_NORM_TYPE 宏与 parallel_for_rows 并行仿函数（design）：三个算符最终共用 DISPATCH_DIFFUSION_NORM_TYPE 与 parallel_for_rows(B, S, D, row_fn)。
- 抽取 load_param_vec2 / apply_residual_gate_vec / apply_norm_modulate_row 三个 helper 消除重复（design）：三个 row kernel 收敛到同一组 helper；gate 缺失时统一按 1.0 处理，避免运行期分支扩散。
- forward_cpu 中强制 .contiguous() 可能引入隐式 memcpy（performance）：最终实现不再对 residual / x 调用 .contiguous()，仅 weight / bias 走 _ensure_contiguous；但 kernel 侧对最后一维连续性的要求依旧存在，非连续输入的实测行为未被测试覆盖。
- 测试参数化写法简化（DTYPE_PAIRS 提取）（testing）：已按建议调整，测试同时覆盖 broadcast_c 的 stride 0 / 1 两条路径与三组真实 gate/norm 组合。

风险与影响

- 风险：1) 全局 CPU 派发语义变更：custom_op.py 中 forward_cpu 默认实现从 return self.forward_cuda(...) 改为 return self.forward_native(...)，且 dispatch_forward() 新增 is_cpu 分支。虽然对未重写 forward_cpu 的算符而言最终行为等价（此前 CPU 走 else 分支到 forward_native），但语义从「CPU 复用 CUDA 实现」变成「CPU 回落 native」，属于对 multimodal_gen 所有 CPU 算符生效的框架级改动，需关注是否有算符其实依赖旧的注释假设。
- 2) 非连续输入的行为边界：kernel 侧要求最后一维连续。评审明确警告过 .contiguous() 会引入隐式 memcpy，最终实现把 residual / x 原样传入，只对 weight / bias 做 _ensure_contiguous。这意味着非最后一维连续的输入（例如来自 unflatten/transpose 的视图）在 CPU 上的行为需要实测确认，测试用例只覆盖了最后一维 stride 0 / 1 两种情况，没有构造非连续输入。
- 3) 数值一致性依赖人工对齐：apply_norm_modulate_row 里刻意做 norm FP32 -> activation dtype -> scale/shift 的降升位（注释「Match CUDA/CuTe activation-dtype boundary」），标量尾循环里同样有一次 static_cast<scalar_t> 往返。参考实现的单测通过 normalized.to(input_dtype).float() 复刻该边界，一旦 CUDA 侧边界调整而 CPU 未同步，两者会静默发散；这是「与 GPU 逐比特对齐」设计带来的长期维护成本。
- 4) dtype 支持面被主动收窄：C++ 签名中 weight / bias 固定为 const float*，gate 只支持 activation dtype 或 FP32 两种；作者表格也确认没有其它组合。若上游模型的 norm 参数是 BF16 而非 FP32，会走到 check 报错而不是静默降级，排障成本落在调用方。
- 5) 测试覆盖不均：test_diffusion_norm.py 只直接覆盖 fused_scale_shift_cpu 与 fused_scale_residual_norm_scale_shift_cpu，fused_norm_scale_shift_cpu 仅通过 layernorm.py 间接生效；expand_scale_shift_cpu_param 的 1D / 2D / 4D（按帧展开）分支

以及 `_fuse_scale_shift_kernel_cpu` 的平台表替换均无独立断言。

6) 性能数据缺少回归门禁: $1.16\times / 1.20\times$ 来自作者单机手工复现, CI 只跑正确性, 没有性能回归检查, 后续若有人改动 `parallel_for_rows` 的并行粒度或 `#pragma GCC unroll 4`, 退化不会被自动发现。 - 影响: 对用户: CPU (Intel x86、ARM64) 上运行 Wan / LingBot / Helios / HunyuanVideo / Hunyuan3D 等 diffusion 模型可获得端到端约 $1.16\times$ 、去噪阶段约 $1.20\times$ 的加速, 且不改变输出语义 (刻意对齐 CUDA/CuTe 数值边界); 对 GPU 用户与 SRT 主线推理完全无影响。对系统: `sgl_kernel` CPU 算子表新增三个定制 operator, `normd.cpp` 成为 diffusion norm 的独立落脚点, 后续同类融合算子可以按同一模板 (模式 tag + traits + `parallel_for_rows` + stride 0/1 归约) 扩展; `CustomOp.forward_cpu` 的默认语义调整则影响 multimodal_gen 框架下所有 CPU 算符的派发路径。对团队: 这是 Intel 在 SGLang 上继 CPU FP8 GEMM (PR#32618) 之后又一条「非 CUDA 平台补齐融合内核」的线, 评审由 maintainer 深度介入并给出可复制的重构范式, 说明 CPU 后端的内核质量门槛已与 GPU 侧趋同。风险主要落在非连续输入与 dtype 收窄这两个尚未被测试覆盖的角落。 - 风险标记: 新增 662 行手写 CPU 内核, CustomOp.forward_cpu 默认语义翻转 (全局影响), 非连续输入未纳入测试覆盖, 混合 dtype 支持面被主动收窄, 数值边界与 CUDA 人工对齐, 存在长期漂移风险, 缺少性能回归门禁

关联脉络

- PR #32618 [CPU] Add fp8_per_tensor_scaled_mm_cpu kernel: 同一模块与技术路线的直接前序: 同样在 python/sglang/kernels/aot/csrc/cpu/ 下新增 CPU 内核、同步修改 torch_extension_cpu.cpp 注册与 test/registered/cpu/ 测试, 并联动 SRT 层的 quantization 分支; 本 PR 复用了这套「C++ 内核 + CPU 单测 + Python 平台分支」的落地范式, 可对照看 CPU 内核注册与测试注册的写法演进。
- PR #39293 [Diffusion] Clean up obsolete worker plumbing, dead helpers, and tests: 同属 multimodal_gen (Diffusion) 子系统, 且都在调整该子系统的分层边界 (一个删遗留 plumbing, 一个新增 CPU 融合路径并修改 CustomOp 派发)。两 PR 合看可以判断 multimodal_gen 的 CPU 支持是新增能力还是清理后的补位。
- PR #35605 [XPU] Use torch scaled_mm for XPU block FP8 linear: 同属「非 CUDA 平台补齐内核并保留 native 回落」的演进线: 该 PR 在 XPU 上切换底层算子并保留 fallback, 本 PR 在 CPU 上把 fuse_scale_shift_kernel 的 cpu 项从 lazy_fallback 换成真实内核, 两者对平台派发表和数值一致性边界的处理方式可直接对比。