PR #25256 完整报告

sgl-project/sglang

[MUSA][Diffusion] Improve wan model inference speed using torch.compile

合并时间：2026-05-17 22:10

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25256

执行摘要

- 一句话：torch.compile 加速 Wan 扩散模型推理
- 推荐动作：该 PR 展示了在扩散模型推理中应用 torch.compile 的典型模式：添加平台回退、新增 forward_xxx 方法、批量装饰原生方法。值得学习其平台分派和条件编译策略。改动虽小但提供清晰 benchmark，适合作为性能优化的参考案例。

功能与动机

提升 Wan 模型在 MUSA 设备（以及 CUDA）上的推理速度，通过 torch.compile 优化关键操作。PR 正文引用：'Improve Wan model inference using torch.compile on MUSA device, also on CUDA.' 并提供了在 S5000 和 H200 上的单步加速数据（分别为 1.09x 和 1.05x）。

实现拆解

1. MUSA 平台回退：在 python/sglang/jit_kernel/diffusion/triton/scale_shift.py 中添加 current_platform.is_musa() 分支，使 fuse_scale_shift_kernel 在 MUSA 上使用 torch_fallback。
2. MulAdd 编译优化：在 python/sglang/multimodal_gen/runtime/layers/elementwise.py 中为 MulAdd 新增 forward_musa 方法，用 @torch.compile 装饰并直接调用 forward_native，使得 MUSA 设备上的 MulAdd 操作被编译优化。
3. Layernorm 编译优化：在 python/sglang/multimodal_gen/runtime/layers/layernorm.py 中对三个 forward_native 方法（_ScaleResidualNormScaleShift.forward_native、_NormScaleShift.forward_native、_NormTanhMulAdd.forward_native）添加 @torch.compile(disable=current_platform.is_npu()) 装饰，避免影响 NPU 路径。
4. 张量连续性保证：在 python/sglang/multimodal_gen/runtime/models/dits/wanvideo.py 中 patch_embedding 后添加 .contiguous() 确保内存连续，满足 torch.compile 要求（虽然后续层也会自动处理，但无额外开销）。
5. 性能验证：提供了 MUSA S5000 和 Nvidia H200 上的 profiler 对比截图，未新增单元测试，但 CI 包含现有集成测试。

关键文件：

- python/sglang/jit_kernel/diffusion/triton/scale_shift.py（模块 JIT 内核；类别 source；类型 dependency-wiring）：添加 MUSA 平台的回退路径，是 MUSA 支持的基础。所有 diffusion 模型中共用的 scale_shift 内核选择逻辑扩展至此。

- python/sglang/multimodal_gen/runtime/layers/elementwise.py（模块 算子层；类别 source；类型 core-logic；符号 forward_musa）：新增 forward_musa 方法，是 torch.compile 在 MulAdd 算子的直接应用入口。
- python/sglang/multimodal_gen/runtime/layers/layernorm.py（模块 算子层；类别 source；类型 core-logic）：对三个核心 Layernorm 变体的 forward_native 启用 torch.compile，影响所有使用这些层的扩散模型。同时通过条件 disable 保护 NPU 路径。
- python/sglang/multimodal_gen/runtime/models/dits/wanvideo.py（模块 模型；类别 source；类型 data-contract）：添加 contiguous 调用确保张量连续，是 torch.compile 的必要条件。虽引发讨论，但最终接受。

关键符号：MulAdd.forward_musa, _ScaleResidualNormScaleShift.forward_native, _NormScaleShift.forward_native, _NormTanhMulAdd.forward_native

关键源码片段

python/sglang/jit_kernel/diffusion/triton/scale_shift.py

添加 MUSA 平台的回退路径，是 MUSA 支持的基础。所有 diffusion 模型中共用的 scale_shift 内核选择逻辑扩展至此。

```
# 在已有平台回退后添加 MUSA 分支
if current_platform.is_npu():
    from .npu_fallback import fuse_scale_shift_native
    fuse_scale_shift_kernel = fuse_scale_shift_native

if current_platform.is_mps():
    from .mps_fallback import fuse_scale_shift_kernel_native
    fuse_scale_shift_kernel = fuse_scale_shift_kernel_native

# 新增：MUSA 平台使用 torch 原生实现（回退）
if current_platform.is_musa():
    from .torch_fallback import fuse_scale_shift_kernel_native
    fuse_scale_shift_kernel = fuse_scale_shift_kernel_native

if current_platform.is_cpu():
    from .torch_fallback import (
        fuse_scale_shift_kernel_native,
    )
    fuse_scale_shift_kernel = fuse_scale_shift_kernel_native
```

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

新增 forward_musa 方法，是 torch.compile 在 MulAdd 算子的直接应用入口。

```
class MulAdd(CustomOp):
    # ... 其他方法 ...

    def forward_xpu(self, a, b, c, k=0):
        return self.forward_native(a, b, c, k=k)
```

```
    # 新增：MUSA 专用编译版本
    @torch.compile
    def forward_musa(
        self, a: torch.Tensor, b: torch.Tensor, c: torch.Tensor, k: int = 0
    ):
        # 直接复用原生实现，torch.compile 负责图优化
        return self.forward_native(a, b, c, k=k)

    def forward_npu(self, a, b, c, k=0):
        from sgl_kernel_npu.norm.scale_shift import fused_scale_shift
        return fused_scale_shift(a, b, c, scale_constant=k)
```

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

对三个核心 Layernorm 变体的 forward_native 启用 torch.compile，影响所有使用这些层的扩散模型。同时通过条件 disable 保护 NPU 路径。

```
class _ScaleResidualNormScaleShift(CustomOp):
    # ... 其他方法 ...

    def forward_xpu(self, *args, **kwargs):
        return self.forward_native(*args, **kwargs)

    # 启用编译，但 NPU 平台除外（disable=True 会跳过编译）
    @torch.compile(disable=current_platform.is_npu())
    def forward_native(
        self,
        residual: torch.Tensor,
        x: torch.Tensor,
        gate: torch.Tensor | int,
        shift: torch.Tensor,
        scale: torch.Tensor,
    ) -> tuple[torch.Tensor, torch.Tensor]:
        # 原有实现：残差连接 + layernorm + scale-shift
        # ...（省略主体，保持不变）
        return modulated, residual_output

class _NormScaleShift(CustomOp):
    @torch.compile(disable=current_platform.is_npu())
    def forward_native(self, x, shift, scale):
        # ...
        return modulated.to(x.dtype)

class _NormTanhMulAdd(CustomOp):
    @torch.compile(disable=current_platform.is_npu())
    def forward_native(self, x, scale, shift):
        # ...
        return result
```

评论区精华

yeahdongcn 在 wanvideo.py 的评论中建议将 .contiguous() 放在 is_musa 条件下，以避免对其他平台产生不必要的内存拷贝。wenqf11 回复展示截图说明即使不加条件，后续的 LayerNormScaleShift 层也会自动做 contiguous，因此添加 contiguity 没有实际影响（no effect）。最终未添加条件控制，PR 经 yeahdongcn 和 mickqian 批准合并。

- contiguous 是否应仅用于 MUSA (design): 未修改，保持无条件 contiguous。PR 经批准合并，说明 reviewer 接受了不影响性能的解释。

风险与影响

- 风险：
  - 性能风险：torch.compile 首次调用有编译开销，适合多步推理；单步加速较小（5-9%），但无负面副作用。
  - 平台兼容：MUSA 使用 torch_fallback，不触发 Triton 内核，正确性有保证但性能提升有限。layernorm 上的 @torch.compile 可通过 disable=is_npu() 排除 NPU，不影响现有 NPU 路径。
  - 内存开销：wanvideo.py 中额外 .contiguous() 增加一次复制，但后续层也会做，不会叠加。
  - 测试覆盖：未针对 torch.compile 编写特定测试，依赖现有集成测试，可能遗漏编译错误或目标平台差异。
- 影响：
  - 用户影响：Wan 模型用户可获得小幅推理加速，无需任何配置变更。
  - 系统影响：对非 MUSA/CUDA 平台无影响；NPU 平台通过条件禁用避免问题。
  - 团队影响：简化了 MUSA 平台支持，为未来其他算子使用 torch.compile 提供了参考模式。
  - 影响范围：所有使用 MulAdd、LayernormScaleShift、RMSNormScaleShift、NormTanhMulAdd 的扩散模型（不限于 Wan）都会在 CUDA/MUSA 上启用 torch.compile，layernorm 的编译优化对 CUDA 也适用（设计上兼容）。
  - 风险标记：平台特定回退，编译开销，无专用测试，NPU 条件禁用

关联脉络

- PR #24732 [codex] Optimize LTX2 split rotary kernel: 同样属于 diffusion 模块的 JIT 内核优化，展示了利用 Triton 和编译器优化的模式。
- PR #25517 [diffusion] feat: configure encoder as layerwise-offload by default: 同一模块（diffusion）的默认配置优化，与本 PR 的性能改进互补。
- PR #25457 [diffusion] add memory-aware component load order: 同样是 diffusion 模块的性能优化 PR，关注内存加载顺序。