PR #37156 完整报告

sgl-project/sglang

[Diffusion] Fuse Qwen-Image FP8 norm and activation quantization

合并时间：2026-08-31 18:19

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/37156

执行摘要

- 一句话：Qwen-Image FP8 norm 激活量化融合，GB300 端到端提速约 21%
- 推荐动作：值得精读。该 PR 是“窄快路径 + 位精确 + 逐项门控”融合工程的高质量样例：1）用 PTX 级对齐解决量化器的 ULP 差异，比“近似一致”的常规做法更严谨，且把两个真实 checkpoint 中点敏感 scale 放进回归矩阵；2）双输出（BF16 + FP8）同时满足性能与生命周期语义，避免了激活内存行为改变带来的隐性问题；3）configure_fp8_norm_quant 的门控集中在 post_load_weights，所有不支持场景静默回退，工程上非常稳健。建议后续融合类 PR 复用这套门控模板与 5+5 平衡基准协议。关注点：无 review 讨论、AMD ROCm CI 失败未解释，合并前建议补一条说明。

功能与动机

该 PR 的动机来自 Baseten 的 Agentic Kernels in Production 一文所描述的“agentic kernels”融合思路：把扩散模型denoise循环中相邻的逐元素算子（norm、AdaLN、静态量化）合并进单个 CUDA kernel，减少 kernel launch 与中间张量读写。Qwen-Image 是 Blackwell 上最高频的 diffusion 模型之一，其注意力与 MLP 入口的 FP8 激活量化（ModelOpt 官方 checkpoint）原本需要先完成 norm + modulation，再单独执行 static_quant_fp8 量化，本 PR 将这两步合而为一，并以 bit-exact 为前提——生成模型对数值扰动敏感，任何 ULP 差异都可能改变输出图像。PR body 用 5+5 平衡进程协议、PNG SHA256 与 SSIM/PSNR/LPIPS 证明了像素级一致。

实现拆解

1. CUDA kernel 层扩展（python/sglang/kernels/jit/csrc/diffusion/norm_scale_shift.cuh）：为现有 norm_scale_shift_kernel 增加模板参数 kQuantizeFp8，在逐元素 norm + scale/shift 之后追加 E4M3 静态量化分支；NormScaleShiftParams 新增 quantized 与 input_scale 字段，并新增 NormScaleShiftFp8Kernel、ScaleResidualNormScaleShiftFp8Kernel 两个入口 struct。量化倒数使用内联 PTX div.full.f32（triton_scale_reciprocal），与 Triton 参考量化器逐字节对齐。
2. JIT 封装与守卫（python/sglang/kernels/ops/diffusion/norm/norm_scale_shift_jit.py）：新增 fused_norm_scale_shift_fp8、fused_scale_residual_norm_scale_shift_fp8（返回 normalized + quantized [+ residual_out]），以及对应的 try_* 守卫版本；守卫逐项校验 norm 类型为 layer、无 affine 权重、输入为连续 BF16、Blackwell 及以上设备、_fp8_input_scale 契约（单元素 FP32 CUDA 张量），任一不满足返回 None 走旧路径。

norm_scale_shift_module() 在非 Blackwell 设备上显式抛 RuntimeError，把硬件依赖显式化。

3. 模型主路径集成（python/sglang/multimodal_gen/runtime/models/dits/qwen_image.py）：QwenImageTransformerBlock.__init__增加 _fp8_img_attn_norm_quant 等 4 个门控标志；新增 _valid_modelopt_fp8_linear（要求 quant_method 为 ModelOptFp8LinearMethod、input_scale 为有限正的单元素 FP32 CUDA 张量）与 _shared_modelopt_fp8_scale（分离 Q/K/V 的 input_scale 必须完全一致）；configure_fp8_norm_quant() 在 post_load_weights 阶段按 fused/ 分离 QKV、added QKV、MLP 入口投影分别开启门控。forward 中新增 _try_fp8_norm_quant 与 _try_fp8_residual_norm_quant：命中融合路径时 FP8 张量作为 attention/MLP 输入，BF16 输出在对应投影完成后 del，维持原激活分配 / 生命周期语义；未命中则走原 _modulate 路径。
4. 导出注册与基准：python/sglang/kernels/ops/diffusion/init.py 注册 4 个新符号（映射到 norm.norm_scale_shift_jit）；新增 bench_qwen_image_norm_fp8_quant.py kernel benchmark，对比 split（norm + 独立 static_quant_fp8）与 fused 两种实现，并在非 Blackwell 上 marker.skip（该 guard 是 PR 内第二个 commit 补上的修复）。
5. 测试配套：新增 test_qwen_image_norm_fp8_quant.py（kernel bit-exact 测试，参数化 rows ∈ {1, 127, 1024}、5 个 scale 值，含两个真实 checkpoint 中点敏感 scale，同时校验 BF16 输出与 FP8 字节完全相等）；新增 test_qwen_image_fp8_norm_quant.py（模型门控单测，覆盖分离 QKV scale 一致 / 不一致、fused QKV、非正 scale 禁用融合三种场景）。

关键文件：

- python/sglang/multimodal_gen/runtime/models/dits/qwen_image.py（模块 模型实现；类别 source；类型 core-logic；符号 _valid_modelopt_fp8_linear, _shared_modelopt_fp8_scale, configure_fp8_norm_quant, _try_fp8_norm_quant）：融合路径的模型侧主入口：新增 4 个 FP8 门控标志、_valid_modelopt_fp8_linear/_shared_modelopt_fp8_scale 校验、configure_fp8_norm_quant 与 _try_fp8_norm_quant/_try_fp8_residual_norm_quant，并在 forward 中把 FP8 张量接入 attention/MLP，BF16 输出延迟释放保持生命周期语义。
- python/sglang/kernels/jit/csrc/diffusion/norm_scale_shift.cuh（模块 CUDA 内核；类别 source；类型 core-logic；符号 NormScaleShiftFp8Kernel, ScaleResidualNormScaleShiftFp8Kernel, triton_scale_reciprocal, NormScaleShiftParams）：CUDA kernel 本体：新增 kQuantizeFp8 模板分支、triton_scale_reciprocal（PTX div.full.f32 对齐 Triton 量化器）以及 NormScaleShiftFp8Kernel / ScaleResidualNormScaleShiftFp8Kernel 两个入口，是本 PR 性能收益的物理来源。
- python/sglang/kernels/ops/diffusion/norm/norm_scale_shift_jit.py（模块 JIT 封装；类别 source；类型 core-logic；符号 fused_norm_scale_shift_fp8, fused_scale_residual_norm_scale_shift_fp8, try_fused_norm_scale_shift_fp8, try_fused_scale_residual_norm_scale_shift_fp8）：JIT 封装与守卫层：定义 fused/try_ 两套 FP8 接口，集中表达快路径的全部前置条件（硬件、dtype、连续性、input_scale 契

约），是融合安全性的第一道闸门。

- test/registered/kernels/ops/diffusion/test_qwen_image_norm_fp8_quant.py（模块 内核测试；类别 test；类型 test-coverage；符号 _make_inputs, test_norm_scale_shift_fp8_is_bit_exact, test_residual_norm_scale_shift_fp8_is_bit_exact）：kernel 级 bit-exact 回归：对 norm 与 residual-norm 两条融合路径，在 1/127/1024 行与 5 个 scale（含两个真实 checkpoint 中点敏感值）下逐一断言 BF16 与 FP8 字节完全相等，是数值精确性承诺的测试载体。
- test/registered/unit/models/test_qwen_image_fp8_norm_quant.py（模块 门控测试；类别 test；类型 test-coverage；符号 _fp8_linear, _attention, _block, TestQwenImageFp8NormQuantGate）：模型门控单测：用 mock 的 ModelOpt FP8 线性层与 SimpleNamespace attention 覆盖 configure_fp8_norm_quant 的三个关键分支（分离 QKV scale 一致才开启、fused QKV 用物化 scale、非正 scale 禁用），防止门控逻辑回归。
- test/registered/kernels/benchmark/diffusion/bench_qwen_image_norm_fp8_quant.py（模块 基准测试；类别 test；类型 benchmark；符号 benchmark, fn）：新增 split vs fused 的 kernel benchmark，覆盖 norm 与 residual-norm 两条路径在 128/1024/4096 行的耗时对比；非 Blackwell 上 marker.skip 的 guard 是 PR 内第二个 commit 补上的 CI 修复。
- python/sglang/kernels/ops/diffusion/__init__.py（模块 内核导出；类别 infra；类型 configuration；符号 fused_norm_scale_shift_fp8, fused_scale_residual_norm_scale_shift_fp8, try_fused_norm_scale_shift_fp8, try_fused_scale_residual_norm_scale_shift_fp8）：把 4 个新符号挂到 diffusion kernels 的懒加载注册表，是 JIT kernel 对模型层可见性的最后一步接线。

关键符号：configure_fp8_norm_quant, _valid_modelopt_fp8_linear, _shared_modelopt_fp8_scale, _try_fp8_norm_quant, _try_fp8_residual_norm_quant, post_load_weights, fused_norm_scale_shift_fp8, fused_scale_residual_norm_scale_shift_fp8, try_fused_norm_scale_shift_fp8, try_fused_scale_residual_norm_scale_shift_fp8, triton_scale_reciprocal, NormScaleShiftFp8Kernel, ScaleResidualNormScaleShiftFp8Kernel

关键源码片段

python/sglang/multimodal_gen/runtime/models/dits/qwen_image.py

融合路径的模型侧主入口：新增 4 个 FP8 门控标志、_valid_modelopt_fp8_linear/_shared_modelopt_fp8_scale 校验、configure_fp8_norm_quant 与 _try_fp8_norm_quant/_try_fp8_residual_norm_quant，并在 forward 中把 FP8 张量接入 attention/MLP，BF16 输出延迟释放保持生命周期语义。

```
  def configure_fp8_norm_quant(self) -> None:
      """在 checkpoint 的 input_scale 物化后，按条件开启 norm+quant 融合路径。"""
      if not torch.cuda.is_available():
          return
      capability = torch.cuda.get_device_capability()
      # 快路径刻意收窄：仅 3072 隐藏维度、Blackwell 及以上（sm_100+）、
```

```
        # 且不启用 zero_cond_t 时间步分支时才可能开启融合。
        if self.dim != 3072 or capability[0] < 10 or self.zero_cond_t:
            return
        if self.attn.use_fused_qkv:
            # 打包 QKV（来自 #37123）场景：直接使用物化后的单一 input_scale。
            self._fp8_img_attn_norm_quant = self._valid_modelopt_fp8_linear(
                self.attn.to_qkv
            )
        else:
            # 分离 Q/K/V 场景：三路投影的 input_scale 必须完全一致，
            # 否则同一个量化结果无法同时满足三路 FP8 消费方，融合保持关闭。
            self._fp8_img_attn_norm_quant = self._shared_modelopt_fp8_scale(
                [self.attn.to_q, self.attn.to_k, self.attn.to_v]
            )
        if self.attn.added_kv_proj_dim is not None:
            # 文本流走 added QKV，门控逻辑与图像流对称。
            if self.attn.use_fused_added_qkv:
                self._fp8_txt_attn_norm_quant = self._valid_modelopt_fp8_linear(
                    self.attn.to_added_qkv
                )
            else:
                self._fp8_txt_attn_norm_quant = self._shared_modelopt_fp8_scale(
                    [self.attn.add_q_proj, self.attn.add_k_proj, self.attn.add_v_proj]
                )
        if isinstance(self.img_mlp, QwenImageFeedForward):
            # MLP 入口投影同样要求是合法的 ModelOpt FP8 线性层。
            self._fp8_img_mlp_norm_quant = self._valid_modelopt_fp8_linear(
                self.img_mlp.net[0].proj
            )
        if isinstance(self.txt_mlp, QwenImageFeedForward):
            self._fp8_txt_mlp_norm_quant = self._valid_modelopt_fp8_linear(
                self.txt_mlp.net[0].proj
            )
```

python/sglang/kernels/jit/csrc/diffusion/norm_scale_shift.cuh

CUDA kernel 本体：新增 kQuantizeFp8 模板分支、triton_scale_reciprocal（PTX div.full.f32 对齐 Triton 量化器）以及 NormScaleShiftFp8Kernel / ScaleResidualNormScaleShiftFp8Kernel 两个入口，是本 PR 性能收益的物理来源。

```
// Triton 的静态 FP8 量化器把 1.0 / scale 下译为 div.full.f32；
// 普通 CUDA 求倒数相差 1 个 FP32 ULP，在 E4M3 中点附近会改变量化字节，
// 因此这里必须使用同一条 PTX 指令以保证 bit-exact。
SGL_DEVICE float triton_scale_reciprocal(float scale) {
    float reciprocal;
    asm("div.full.f32 %0, %1, %2;" : "=f"(reciprocal) : "f"(1.0f), "f"(scale));
    return reciprocal;
}

// kQuantizeFp8 分支：先得到 BF16 的 norm+scale/shift 结果，
```

```
// 再乘 input_scale 倒数并 clamp 到 E4M3 表示范围，写出 FP8 张量。
// BF16 与 FP8 双输出同时保留：FP8 直接供 ModelOpt 投影消费，
// BF16 继续存活以维持原有激活分配 / 生命周期语义。
const float input_scale_inv = triton_scale_reciprocal(
    *static_cast<const float*>(params.input_scale));
#pragma unroll
for (int i = 0; i < kVecElems; ++i) {
    const float norm = static_cast<float>(static_cast<bf16_t>((v[i] - mean) * factor));
    const bf16_t rounded = static_cast<bf16_t>(
        norm * (1.0f + static_cast<float>(scv[i])) + static_cast<float>(shv[i]));
    yv[i] = rounded;
    if constexpr (kQuantizeFp8) {
        const float scaled = static_cast<float>(rounded) * input_scale_inv;
        const float clamped = math::min(
            math::max(scaled, -DTypeTrait<fp8_e4m3_t>::kFloatMax),
            DTypeTrait<fp8_e4m3_t>::kFloatMax);
        qv[i] = static_cast<fp8_e4m3_t>(clamped);
    }
}
```

python/sglang/kernels/ops/diffusion/norm/norm_scale_shift_jit.py

JIT 封装与守卫层：定义 fused/try_ 两套 FP8 接口，集中表达快路径的全部前置条件（硬件、dtype、连续性、input_scale 契约），是融合安全性的第一道闸门。

```
def fused_norm_scale_shift_fp8(x, scale, shift, input_scale, eps):
    """单次 kernel 同时产出 BF16 modulation 输出与静态 E4M3 量化结果。"""
    normalized = torch.empty_like(x)
    quantized = torch.empty_like(x, dtype=torch.float8_e4m3fn)
    _module().nss_fp8_row(
        normalized.view(-1, _HIDDEN),
        quantized.view(-1, _HIDDEN),
        x.view(-1, _HIDDEN),
        scale,
        shift,
        input_scale.reshape(1),
        float(eps),
    )
    return normalized, quantized


def try_fused_norm_scale_shift_fp8(x, weight, bias, scale, shift, input_scale, norm_type, eps):
    # 门控守卫：仅 LayerNorm、无 affine 权重、输入为连续 BF16、运行在
    # Blackwell 及以上、input_scale 为单元素 FP32 CUDA 张量时才融合；
    # 任何一个条件不满足都返回 None，由调用方走原有路径。
    if norm_type != "layer" or weight is not None or bias is not None:
        return None
    if not _nss_activation(x) or not _blackwell_or_newer(x.device):
        return None
    scale = _row_bf16(scale, x.device)
```

```
    shift = _row_bf16(shift, x.device)
    if scale is None or shift is None:
        return None
    if not _fp8_input_scale(input_scale, x.device):
        return None
    return fused_norm_scale_shift_fp8(x, scale, shift, input_scale, eps)
```

评论区精华

本 PR 没有任何 review 评论或讨论线程（comments_count 与 review_comments_count 均为 0），由作者 BBuf 直接合并，关键设计取舍以 PR body 形式沉淀。其中最重要的数值精确性决策为：Triton 参考静态量化器把 1.0 / scale 下译为 PTX div.full.f32，普通 CUDA 求倒数会差 1 个 FP32 ULP，对真实 checkpoint scale（如 0.4754464328）可能把数值推过 E4M3 中点而改变量化字节，因此融合 kernel 内联同一条 PTX 指令，并把两个 midpoint-sensitive scale 纳入回归矩阵，BF16 输出与 FP8 字节都做完全相等校验。此外，快路径范围被刻意收窄（Blackwell+、BF16 batch-1 连续张量、hidden 3072、affine-free LayerNorm、有限正标量 FP32 input_scale、无 indexed modulation、无 zero-condition-timestep 路径、不可断 CUDA graph），每个不支持场景都保留旧路径。基准协议也值得记录：5 个独立进程、交替顺序 main/PR、GPU 锁频 1800 MHz，用 PNG SHA256 与 SSIM/PSNR/LPIPS 验证像素级一致。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. BF16 生命周期依赖 del 顺序：forward 在 joint attention 之后立即 del img_modulated_bf16、txt_modulated_bf16，当前代码不再引用这两个张量，但任何后续改动若在 attention 之后读取 BF16 modulation 输出（调试钩子、新融合、梯度检查点），会触发使用已释放张量或静默语义变化。
  2. 静默回退风险：configure_fp8_norm_quant 依赖 checkpoint 加载后 input_scale 已物化且带 ModelOptFp8LinearMethod；若未来加载时序变化（lazy loading、offload、transformer 替换检测逻辑调整），所有门控会静默保持 False 退化为旧路径——功能安全，但性能回归不易察觉，需要 benchmark 基线监控。
  3. 数值一致性依赖：bit-exact 依赖 Triton 参考量化器仍把倒数下译为 div.full.f32；若 static_quant_fp8 或 ModelOpt 投影实现变化（依赖升级、切换 quant 后端），字节一致性可能被破坏。测试矩阵覆盖了真实 checkpoint scale，但未覆盖 FP16 input_scale 或非 ModelOpt 量化后端。
  4. 硬件范围与 CI：融合仅限 CUDA Blackwell+，norm_scale_shift_module() 在非 Blackwell 上抛 RuntimeError，依赖 try_* 守卫保证不被误调；CPU/XPU/ROCm 环境不受影响，但 AMD ROCm 7.2 CI 失败未在 PR 中说明（PR 内容与 AMD 无关，仍应关注是否为环境问题）。
  5. 性能反模式边界：fused kernel 同时写 BF16 与 FP8 双输出，比纯 FP8 路径多一趟 BF16 写回；microbenchmark 显示 1.1~1.2x 融合收益，但该收益依赖 hidden 3072

且行数在 128~4096 范围，更大 batch 或其它模型尺寸不适用（kHidden 与 _HIDDEN 为硬编码约束）。 - 影响：用户侧：Qwen-Image + ModelOpt FP8 官方 checkpoint 用户在 GB300（以及 B200）上获得约 21% 端到端提速（denoise/step 均值 306.1 →240.9 ms），图像输出与 main 像素级一致；非 Blackwell、非 FP8、非 Qwen-Image用户完全无感（门控关闭走旧路径）。系统侧：diffusion kernels 新增 2 个 CUDAkernel 入口、4 个导出符号，JIT 模块新增 2 个 kernel 运行符，符号注册表同步扩展；kernel 单元测试注册到 4-gpu-b200 网格。团队侧：与 #37123（fused FP8 QKV）叠加后 denoise/step 进一步降至约 201 ms（再降 18%），形成 Qwen-Image BlackwellFP8 融合系列；同仓库近期的 Qwen-Image adaLN 融合（#37144）、bias 吸收（#37116）共用 qwen_image.py 与 norm_scale_shift.cuh，文件冲突面在扩大，需要更细的模块划分。 - 风险标记：快路径门控复杂，Blackwell 专属硬件路径，PTX 级数值一致性依赖，AMD ROCm CI 失败未说明，BF16 生命周期依赖 del 顺序

关联脉络

- PR #37123 Fused FP8 QKV（PR body 引用，未在历史列表）: PR body 明确说明 packed QKV 来自 #37123，其 materialized packed scale 被本 PR 的 configure_fp8_norm_quant 直接消费；PR 实测数据也给出了与 #37123 叠加后的增量收益（约 18%）。
- PR #37144 [Diffusion] Fuse Qwen-Image final adaptive LayerNorm: 同一 qwen_image.py 的同类融合，把 final adaLN 并入 fused kernel；两者共享 norm_scale_shift.cuh 与 bit-exact 回归体系，属于同一融合家族。
- PR #37116 [diffusion] perf: absorb Qwen-Image output projection biases: 同一 Qwen-Image 性能融合系列，同样改动 qwen_image.py、norm_scale_shift.cuh、JIT 注册与 bench，文件冲突面与演进方向一致。
- PR #37141 [Diffusion] Fuse FLUX.2 token concatenation and NVFP4 quantization: 把“融合 + 量化”策略应用到另一 diffusion 模型（FLUX.2），共享 model_fast_paths 测试框架与类似的门控 / 回退设计。
- PR #36991 [Diffusion] Add exact component precision overrides: 组件级精度 / 量化配置基础设施，ModelOpt FP8 的 input_scale 契约与量化方法识别逻辑来自该体系。
- PR #36916 [Diffusion] Detect quantized transformer replacements: 从替换权重自身探测量化声明，与本 PR 的 _valid_modelopt_fp8_linear 识别 ModelOpt FP8 线性层的方法一脉相承。