PR #24897 完整报告

sgl-project/sglang

Port fused SiLU+clamp+FP8 quant from DSV4 dev branch

合并时间：2026-05-13 22:36

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24897

执行摘要

- 一句话：为 DeepseekV2MLP 添加 fused SiLU+clamp+FP8 量化路径
- 推荐动作：值得精读，尤其是 fused 路径的条件判断和与 deep_gemm 的集成方式；注意 review 中关于 transposed 参数命名的讨论，建议后续完善注释。

功能与动机

从 PR body 和提交信息可知，本 PR 旨在将 DSV4 dev 分支的 fused SiLU+clamp+FP8 量化优化移植到主分支，通过减少 HBM 读写次数提升推理性能。此外，W_o 路径的 fast_fp8_quant Triton 内核被 revert，改用已在 #24628 中得到验证的更快的 sglang_per_token_group_quant_fp8 v2 CUDA 内核。

实现拆解

1. 导入新增内核：在 deepseek_v2.py 文件头部添加 from sglang.jit_kernel.deepseek_v4 import silu_and_mul_clamp, silu_and_mul_contig_post_quant 以及 from sglang.srt.layers.quantization.fp8_kernel import create_per_token_group_quant_fp8_output_scale，为 fused 操作准备。
2. 新增 fast path 判定条件：在 DeepseekV2MLP.forward 中，当 self.swiglu_limit is not None、down_proj.reduce_results 为假、down_proj.weight 为 uint8 类型且存在 weight_scale_inv 属性时，尝试执行 fused 路径。
3. fused 路径核心逻辑：创建 FP8 输出张量（float8_e4m3fn）和对应的 scale 张量，调用 silu_and_mul_contig_post_quant 一次完成激活、clamp 和 per-token-group 量化，然后通过 deep_gemm_wrapper.gemm_nt_f8f8bf16 直接执行 FP8 GEMM 得到 BF16 结果，全程避免 BF16 中间张量的 HBM 写入。
4. 回退路径优化：原有的 silu_and_mul 前对 gate_up 手动 clamp 并 cat 的方式被替换为 silu_and_mul_clamp 融合内核，减少 kernel launch 和 HBM 访问；当不需要 swiglu_limit 时维持原 act_fn 调用。
5. 移除废弃导入：清理不再使用的 fast_fp8_quant 模块导入（对应 commit 7c354f3）。

关键文件：

- python/sglang/srt/models/deepseek_v2.py（模块 模型实现；类别 source；类型 core-logic；符号 DeepseekV2MLP.forward）：唯一修改文件，核心 MLP 前向方法

关键符号：DeepseekV2MLP.forward

关键源码片段

python/sglang/srt/models/deepseek_v2.py

唯一修改文件，核心 MLP 前向方法

```
# python/sglang/srt/models/deepseek_v2.py

# ... 在 DeepseekV2MLP.forward 方法中
def forward(self, x):
    # ... 前置计算 gate_up ...
    gate_up, _ = self.gate_up_proj(x)

    # Fast path: fused SiLU + clamp + per-token-group FP8 quant + deep_gemm
    # 仅在满足以下条件时启用：
    # - swiglu_limit 不为 None（需要 clamp）
    # - down_proj 不需要 all-reduce
    # - down_proj 权重为 uint8（FP8 量化存储）且包含 weight_scale_inv
    if (
        self.swiglu_limit is not None
        and not self.down_proj.reduce_results
        and self.down_proj.weight.dtype == torch.uint8
        and hasattr(self.down_proj, "weight_scale_inv")
    ):
        M, N = gate_up.shape
        # FP8 输出（半宽，因为 gate_up 是 gate 和 up 拼接）
        down_input_fp8 = gate_up.new_empty((M, N // 2), dtype=torch.float8_e4m3fn)
        scale_block_size = 128
        # 创建 per-token-group 量化 scale 张量
        down_input_scale = create_per_token_group_quant_fp8_output_scale(
            x_shape=(M, N // 2),
            device=gate_up.device,
            group_size=scale_block_size,
            column_major_scales=deep_gemm_wrapper.DEEPGEMM_SCALE_UE8M0,
            scale_tma_aligned=deep_gemm_wrapper.DEEPGEMM_SCALE_UE8M0,
            scale_ue8m0=deep_gemm_wrapper.DEEPGEMM_SCALE_UE8M0,
        )
        # 融合内核：silu + clamp + per-group fp8 quant，结果写入 down_input_fp8
        silu_and_mul_contig_post_quant(
            input=gate_up,
            output=down_input_fp8,
            output_scale=down_input_scale,
            quant_group_size=scale_block_size,
            scale_ue8m0=deep_gemm_wrapper.DEEPGEMM_SCALE_UE8M0,
            transposed=deep_gemm_wrapper.DEEPGEMM_SCALE_UE8M0, # 实际表示 Blackwell
            swizzled 布局，非 tensor 转置
            swiglu_limit=float(self.swiglu_limit),
        )
```

```
        # 直接使用 deep_gemm 的 FP8 GEMM，输出 BF16
        down_output = gate_up.new_empty(
            (M, self.down_proj.output_size), dtype=torch.bfloat16
        )
        deep_gemm_wrapper.gemm_nt_f8f8bf16(
            (down_input_fp8, down_input_scale),
            (self.down_proj.weight, self.down_proj.weight_scale_inv),
            down_output,
        )
        return down_output

    # Fallback: fused silu+clamp kernel（仍比原来 unfused 的 chunk+clamp+cat 更快）
    if self.swiglu_limit is not None:
        M, N = gate_up.shape
        x = gate_up.new_empty((M, N // 2))
        silu_and_mul_clamp(gate_up, x, float(self.swiglu_limit))
    else:
        x = self.act_fn(gate_up)

    x, _ = self.down_proj(x, skip_all_reduce=...)
    return x
```

评论区精华

1. linear_bf16_fp32 auto selector 是否合入：DarkSharpness 评论认为该选择器没有明显的性能收益，建议去掉该部分；yhyang201 询问是否丢弃整个模块。最终该部分并未包含在本次 PR 中（仅 deepseek_v2.py 变更）。
2. fused 内核中 transposed 参数命名困惑：gemini-code-assist[bot] 指出将 DEEPGEMM_SCALE_UE8M0 传递给 transposed 参数令人困惑，建议更名或增加注释说明其表示 Blackwell swizzled 布局。该建议未被采纳，PR 已合并。
3. W_o 路径的 fast_fp8_quant 被 v2 CUDA 内核取代：zcnrex 引用 #24628 说明新 CUDA v2 内核更快，yhyang201 立即回退，使用更快的 sglang_per_token_group_quant_fp8 v2。

- linear_bf16_fp32 auto selector 是否合入 (performance): 该部分未包含在最终提交中（仅 deepseek_v2.py 变更）
- fused 内核中 transposed 参数命名困惑 (design): 未明确采纳，PR 已合并
- W_o 路径的 fast_fp8_quant 被 v2 CUDA 内核取代 (performance): 最终 revert 了 fast_fp8_quant Triton 内核，恢复为使用 sglang_per_token_group_quant_fp8 v2

风险与影响

- 风险：
  1. deep_gemm 依赖：fast path 需要 deep_gemm 可用，若环境未安装将导致运行时错误；但该依赖已在 DSV4 分支中使用，风险可控。

2. 量化对齐：fused 内核中 scale_block_size 与 deep_gemm 期望的对齐方式必须一致，否则产生错误；当前硬编码为 128，需确保与模型配置匹配。
3. 条件覆盖不足：fast path 的条件判断依赖 swiglu_limit、权重类型等，若模型配置不满足条件则走回退路径，可能隐藏未测试的交互。
4. 回退路径行为变化：回退路径从手动 clamp+cat 改为 fused 内核，对于非 FP8 权重的模型可能引入差异，需要验证。 - 影响：直接影响使用 DeepSeek-V2/V4 模型的用户，MLP 前向性能提升（减少 HBM 写入），对 FP8 权重配置收益最大。无 API 或配置变更，向后兼容。团队需确保 deep_gemm 在部署环境可用。 - 风险标记：核心路径变更，缺少测试覆盖

关联脉络

- PR #24628 启用 sglang_per_token_group_quant_fp8 v2 CUDA kernel 提高性能：本 PR 引用了该 PR 的改进，用 v2 替换了 Triton 版 fast_fp8_quant
- PR #24890 Port KV Compression V2 from deepseek_v4_dev: 同系列从 DSV4 分支移植的 PR，且共享部分 jit kernel 依赖