PR #35593 完整报告

sgl-project/sglang

[Fix] Support 128-aligned hidden sizes in the W4AFP8 DeepEP low-latency requant kernel

合并时间：2026-08-20 13:18

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35593

执行摘要

- 一句话：允许 W4AFP8 requant 内核处理仅 128 对齐的隐藏维度
- 推荐动作：值得精读。三个设计点尤其值得学习：一是用 HAS_K_TAIL: tl.constexpr 做编译期特化，让完整块路径零开销、尾部路径单独获得掩码；二是测试通过整数输入 + 2 的幂 scale 实现 e4m3 位精确断言，并用哨兵值验证 padding 行不被触碰，测试设计严谨且可复现；三是性能对比方法论（CUDA graph 内计时排除 Python 启动开销、A/B 顺序交错避免首次测量偏差、opcode 与访存宽度对比证明无向量化损失），可直接迁移到其他 Triton 内核变更的评审中。

功能与动机

PR body 明确指出，fp8_per_token_to_per_tensor_quant_triton() 原先要求 x.size(2) % K_BLOCK_SIZE == 0（K_BLOCK_SIZE = 1024），因此拒绝任何仅 128 对齐的专家 hidden 尺寸（例如 3584）。由于 W4AFp8MoEMethod 将 low-latency 线格式固定为 fp8，且 apply_deepep_ll() 拒绝不带 per-token-group scales 的载荷，这类模型"完全没有可用的 low-latency 路径"，decode CUDA-graph 捕获时会直接触发断言。作者分析指出 1024 是这条路径上唯一的非必要约束：w1_scale 的 512 对齐是真实硬性要求，而 DeepEP low-latency 的 fp8 scales 形状为 [E, M, hidden // 128]，因此 128 对齐即可安全支持。

实现拆解

1. 变更入口与网格放宽：python/sglang/kernels/ops/moe/ep_moe_kernels.py 中 fp8_per_token_to_per_tensor_quant_triton() 将启动网格从精确除法 x.size(2) // K_BLOCK_SIZE 改为 triton.cdiv，并删除 assert x.size(2) % K_BLOCK_SIZE == 0。这是整条路径中唯一限制 hidden 必须 1024 整块的约束，删除后 128 对齐（如 3584，即 7 × 512、28 个 scale group）即可通过。
2. 内核掩码特化：_fp8_per_token_quant_to_per_tensor_quant_kernel 新增 HAS_K_TAIL: tl.constexpr 常量参数。当其为真（hidden % 1024 != 0）时，对 hidden 加载、scale 加载与 store 统一应用 k_mask = k_offsets < k 掩码；为假时保持完全未掩码的原始加载与相同的启动几何，确保既有 hidden 尺寸的 codegen 与旧版完全一致。这镜像了同文件中 _silu_and_mul_post_per_tensor_quant_kernel 已有的 cdiv 网格 + 掩码快速轴模式。
3. scale 布局兼容性：DeepEP low-latency 的 per-token-group fp8 scale 形状为 [E, M, hidden // 128]，最后两维列主序（TMA 友好），因此由 (k_offsets // K_SCALE_BLOCK_SIZE) * x_scale_stride2 推导的 scale 偏移在尾部块天然落在合法范围

内，只需随 k_mask 一并保护读取边界即可；other=0.0 的掩码值经反量化乘法后仍为 0，且不会写回输出。

4. 单元测试配套：新增 test/registered/kernels/ops/moe/test_fp8_per_token_to_per_tensor_quant.py，参数化 k ∈ {7168, 3584, 1152}（一个完整块、两个尾部尺寸），masked_m 取 {0, 1, 17, m} 验证 masked_m 之后的 padding 行严格保持调用方写入的哨兵值；输入刻意选用 [-8, 8] 整数且 group scale 为 2 的幂，保证每个中间值在 e4m3 中精确可表示，从而支持 rtol=0, atol=0 的位精确断言。测试登记在 base-b-kernel-unit / 1-gpu-large CI stage。
5. 性能与正确性验证：作者在 H200 与 B200 上对既有 shapes 做 CUDA graph 内计时，相对旧内核波动在 -1.66% ~ +1.46% 之间；codegen 对比显示 opcode 集合相同、访存宽度不变（warps=2 → v4.b32、warps=4 → v2.b32、warps=8 → 标量 b32），证明掩码不损失向量化。新增 shape k=3584、8 experts、1024 live rows 时为 H200 25.4 µs / B200 14.6 µs。

关键文件：

- python/sglang/kernels/ops/moe/ep_moe_kernels.py（模块 MoE 内核；类别 source；类型 core-logic；符号 _fp8_per_token_quant_to_per_tensor_quant_kernel, fp8_per_token_to_per_tensor_quant_triton）：核心修复文件：移除 1024 对齐断言、网格改 cdiv、新增 HAS_K_TAIL 掩码特化，是解锁 W4AFP8 DeepEP low-latency 路径的关键。
- test/registered/kernels/ops/moe/test_fp8_per_token_to_per_tensor_quant.py（模块 MoE 内核；类别 test；类型 test-coverage；符号 _build, _ref, test_masked_rows_and_k_tail）：新增参数化单元测试，覆盖精确整块（7168）与两个尾部尺寸（3584、1152），并通过哨兵值验证 masked_m 之后的 padding 行不被改写；输入刻意选择整数与 2 的幂 scale 支持 bit-exact 断言。

关键符号：_fp8_per_token_quant_to_per_tensor_quant_kernel, fp8_per_token_to_per_tensor_quant_triton, _build, _ref, test_masked_rows_and_k_tail

关键源码片段

python/sglang/kernels/ops/moe/ep_moe_kernels.py

核心修复文件：移除 1024 对齐断言、网格改 cdiv、新增 HAS_K_TAIL 掩码特化，是解锁 W4AFP8 DeepEP low-latency 路径的关键。

```
# python/sglang/kernels/ops/moe/ep_moe_kernels.py（按 PR 变更整理的完整函数体）
@triton.jit
def _fp8_per_token_quant_to_per_tensor_quant_kernel(
    x_ptr, x_scale_ptr, output_ptr, output_scale_ptr,
    m, k,
    K_SCALE_BLOCK_SIZE: tl.constexpr,
    K_BLOCK_SIZE: tl.constexpr,
    HAS_K_TAIL: tl.constexpr, # True 表示最后一个 k 块不完整（hidden 仅 128 对齐）
):
    # pid 布局：(k 块，m 块，expert)；m 块的 token 范围由 masked_m 映射而来，
    # 此处省略与旧版一致的 pid → token_id / pid_m_dim 换算细节。
```

```
    pid_k, pid_m, pid_e = (tl.program_id(axis=0), tl.program_id(axis=1), tl.program_id(axis=2))
    output_scale_val_inv = 1.0 / tl.load(output_scale_ptr).to(tl.float32)

    k_offsets = pid_k * K_BLOCK_SIZE + tl.arange(0, K_BLOCK_SIZE)
    # 尾部块只访问 k 以内的通道；完整块走未掩码路径，保证 codegen 与旧版完全一致
    if HAS_K_TAIL:
        k_mask = k_offsets < k

    # DeepEP low-latency 的 per-token-group scale 是列主序（TMA 友好），
    # 所以 scale 偏移也要按 k_offsets 换算，并随 k_mask 一并保护读取边界。
    scale_offsets = (k_offsets // K_SCALE_BLOCK_SIZE) * x_scale_stride2
    x_ptrs = x_ptr + pid_e * m * k + k_offsets
    output_ptrs = output_ptr + pid_e * m * k + k_offsets
    x_scale_ptrs = x_scale_ptr + pid_e * x_scale_stride0 + scale_offsets

    for tok_idx in tl.range(token_id, last_effective_id, pid_m_dim):
        if HAS_K_TAIL:
            hidden = tl.load(x_ptrs + tok_idx * k, mask=k_mask, other=0.0)
            x_scale = tl.load(x_scale_ptrs + tok_idx * x_scale_stride1, mask=k_mask, other=0.0)
        else:
            hidden = tl.load(x_ptrs + tok_idx * k)
            x_scale = tl.load(x_scale_ptrs + tok_idx * x_scale_stride1)

        hidden = hidden.to(tl.float32)
        scale_fp32 = x_scale.to(tl.float32)
        # 反量化：per-token-group scale 乘 per-tensor 输出 scale 的倒数
        hidden = hidden * scale_fp32 * output_scale_val_inv
        # cvt.rn.satfinite 饱和转换：fp8 溢出时钳到 448.0，而不是产生 NaN
        quantized = hidden.to(output_ptr.dtype.element_ty)

        if HAS_K_TAIL:
            tl.store(output_ptrs + tok_idx * k, quantized, mask=k_mask)
        else:
            tl.store(output_ptrs + tok_idx * k, quantized)


def fp8_per_token_to_per_tensor_quant_triton(x, x_scale, masked_m, output_scale, output):
    """per-token fp8 → per-tensor fp8 重量化（W4AFP8 DeepEP low-latency 路径）。"""
    # ... 前置 shape 校验与 masked_m 对应的 m 计算（与旧版一致）...
    K_BLOCK_SIZE = 1024
    # 原来：assert x.size(2) % K_BLOCK_SIZE == 0; grid = (x.size(2) // K_BLOCK_SIZE, 32, x.
    size(0))
    # 现在：用 cdiv 兼容 128 对齐但非 1024 整块的 hidden 尺寸
    grid = (triton.cdiv(x.size(2), K_BLOCK_SIZE), 32, x.size(0))
    _fp8_per_token_quant_to_per_tensor_quant_kernel[grid](
        x, x_scale, output, output_scale, x.size(1), x.size(2),
        K_SCALE_BLOCK_SIZE=K_SCALE_BLOCK_SIZE,
        K_BLOCK_SIZE=K_BLOCK_SIZE,
        HAS_K_TAIL=x.size(2) % K_BLOCK_SIZE != 0,
```

```
        num_warps=8,
    )
```

test/registered/kernels/ops/moe/test_fp8_per_token_to_per_tensor_quant.py

新增参数化单元测试，覆盖精确整块（7168）与两个尾部尺寸（3584、1152），并通过哨兵值验证 masked_m 之后的 padding 行不被改写；输入刻意选择整数与 2 的幂 scale 支持 bit-exact 断言。

```
"""Unit test for ``fp8_per_token_to_per_tensor_quant_triton`` across hidden sizes.

W4AFP8 DeepEP low-latency requantizes the fp8 dispatch payload with this kernel
before the first CUTLASS grouped GEMM.  The payload's hidden size is only
guaranteed to be a multiple of the fp8 scale-group size (128) -- e.g. 3584 for
Kimi-K3 -- so the kernel must handle a ``k`` tail that does not fill a whole
``K_BLOCK_SIZE`` (1024) block, and must still leave the rows past ``masked_m``
untouched.
"""

import pytest
import torch

from sglang.kernels.ops.moe.ep_moe_kernels import (
    fp8_per_token_to_per_tensor_quant_triton,
)
from sglang.test.ci.ci_register import register_cuda_ci

register_cuda_ci(est_time=20, stage="base-b-kernel-unit", runner_config="1-gpu-large")

dev = "cuda"
FP8 = torch.float8_e4m3fn
K_SCALE_BLOCK_SIZE = 128
# 哨兵值 0.375：内核能产生的所有值都是 0.25 的倍数，
# 因此若内核误写 padding 行，必然与哨兵值不等，测试即可捕获。
SENTINEL = 0.375
OUTPUT_SCALE = 2.0


def _build(num_experts, m, k, seed):
    g = torch.Generator(device="cpu").manual_seed(seed)
    # 整数输入 [-8, 8] + 2 的幂 per-token-group scale，保证每个中间值
    # 在 e4m3 中精确可表示，参考实现可与内核 bit-for-bit 一致。
    x = torch.randint(-8, 9, (num_experts, m, k), generator=g).float()
    exps = torch.randint(-1, 2, (num_experts, m, k // K_SCALE_BLOCK_SIZE), generator=g)
    x_scale = torch.pow(2.0, exps.float())
    return x.to(dev).to(FP8), x_scale.to(dev)


def _ref(x, x_scale):
```

```
    # torch 参考：先反量化，再统一乘 per-tensor 输出 scale 的倒数
    dequant = x.float() * x_scale.repeat_interleave(K_SCALE_BLOCK_SIZE, dim=2)
    return (dequant * (1.0 / OUTPUT_SCALE)).to(FP8)


# 7168: K_BLOCK_SIZE 的精确整倍（DeepSeek-V3 的 hidden 尺寸）。
# 3584 / 1152: 仅 128 对齐，最后一个 k 块被部分掩码。
@pytest.mark.parametrize("k", [7168, 3584, 1152])
def test_masked_rows_and_k_tail(k):
    num_experts, m = 4, 48
    masked = [0, 1, 17, m]

    x, x_scale = _build(num_experts, m, k, seed=k)
    masked_m = torch.tensor(masked, dtype=torch.int32, device=dev)
    output_scale = torch.tensor([OUTPUT_SCALE], dtype=torch.float32, device=dev)
    # 用哨兵值填充输出，以便检测内核是否越界写 padding 行
    output = torch.full((num_experts, m, k), SENTINEL, device=dev).to(FP8)

    fp8_per_token_to_per_tensor_quant_triton(
        x=x,
        x_scale=x_scale,
        masked_m=masked_m,
        output_scale=output_scale,
        output=output,
    )

    ref = _ref(x, x_scale)
    for e, valid in enumerate(masked):
        # 有效行必须与 torch 参考位精确一致（rtol=0, atol=0）
        torch.testing.assert_close(
            output[e, :valid].float(), ref[e, :valid].float(), rtol=0, atol=0
        )
        # padding 行不属于任何 expert 的 GEMM 问题规模，必须保持调用方写入的原值
        padding = output[e, valid:].float()
        torch.testing.assert_close(
            padding, torch.full_like(padding, SENTINEL), rtol=0, atol=0
        )


if __name__ == "__main__":
    import sys

    sys.exit(pytest.main([__file__, "-v", "-s"]))
```

评论区精华

该 PR 没有公开的 review 评论线程，唯一审核人 BBuf 直接 APPROVED（无批注）。核心论证全部集中在 PR body 中：作者用充分的证据链说明了 1024 是唯一非必要约束、512 对齐与 128 对齐是真正的硬性边界，并通过 codegen 对比、双卡三 warp 配置的时序矩阵以及 A/B

顺序交错的方法论，自证了“既有 shape 零回归、尾部路径不损失向量化”。虽然没有评审交锋，但 body 本身的论证质量值得作为内核改动评审的范本。

- 暂无高价值评论线程

风险与影响

- 风险：
  - 端到端验证空白：作者明确声明未覆盖“多节点端到端 decode 运行”，内核层面解锁不等于 DeepEP low-latency 通信 + CUTLASS W4A8 grouped GEMM 整条路径在真实模型（如 Kimi-K3）上验证完毕，这是合并后最需要补的闭环。
  - fp8 溢出语义差异：内核使用 cvt.rn.satfinite 饱和转换（钳到 448.0），而 Tensor.to(torch.float8_e4m3fn) 产生 NaN，因此测试刻意将输入限制在安全范围内；真实数据分布若接近 fp8 上界，内核输出与 torch 参考不可比，需以模型准确率评估为准。
  - 尾部掩码路径新增分支：HAS_K_TAIL=True 的载荷 / 存储 /scale 三重掩码是首次上线的新代码路径，虽然 codegen 与向量化检查通过，但 B200 num_warps=4 的小 shape 出现 +1.11% / +1.46% 的正向计时波动，量级小但仍建议在更大形状矩阵上复核。
  - JIT 缓存变体增加：特化会多产生一份尾部版内核变体，编译缓存略增，影响可忽略。
- 影响：
  - 用户侧：hidden 仅 128 对齐的模型（如 Kimi-K3 的 3584）现在可以启用 W4AFP8 + DeepEP low-latency 推理，此前只能退回 per-token 路径或放弃量化，量化部署选项显著扩大。
  - 系统侧：只有尾部尺寸模型启动时会编译新的掩码变体；已支持的 1024 整块尺寸走完全相同的未掩码路径，codegen 无变化，无回归面。
  - 团队侧：新增一个 kernel 单元测试，登记在 base-b-kernel-unit / 1-gpu-large CI stage，预计增加约 20 秒 CI 耗时；无文档或用户可见行为变化。
  - 风险标记：端到端多节点验证待补，fp8 饱和与 torch NaN 语义差异，尾部掩码新分支

关联脉络

- PR #35372 [Kernel] Support wider rows in mega_moe_pre_dispatch: 同仓库同类工作：放宽 MoE 内核的维度对齐 / 宽度约束（本次是 hidden 1024 整块约束），两者都作用于 DeepSeek/DeepEP MoE 内核路径，体现“逐步放宽内核约束”的演进脉络。
- PR #32327 [DeepSeek-V4] Add Q8KV8 sparse MLA prefill runtime backend: 同为 DeepSeek 量化内核路径的功能扩展，涉及 quant/jit-kernel 标签，与本 PR 同属 DeepSeek 系列量化部署方向的拼图。
- PR #35571 [sampling] Fix int32 offset overflow in top-k renorm Triton kernels: 同为 Triton 内核边界条件修复（偏移计算越界），修复模式相似，都是通过掩码或类型调整消除边界错误。