PR #32755 完整报告

sgl-project/sglang

[Perf] Occupancy tuning for DSA indexer fp8-quant Q kernel

合并时间：2026-08-13 16:15

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/32755

执行摘要

- 一句话：索引器量化 Q 内核扩至 256 线程，大 batch 提速约 22%
- 推荐动作：建议精读。虽然最终 diff 只有一行常量，但评审过程包含高价值的性能工程方法论：用 NCU 定位调度瓶颈、用消融实验否决复杂方案（persistent grid-stride 反而更慢）、把改动收敛到最小有效单元。测试文件的 torch reference 写法（含 fp8 反量化容限函数 _fp8_dequant_ok）可作为同类量化内核测试的模板；同时注意测试文件注释与最终实现存在 grid-stride 残留描述，阅读时需对照 PR 演进历史理解。

功能与动机

该内核是 DSA C4 索引器的 fp8-quant Q 内核，被 V4（rope-hadamard 路径）和 V3.2/GLM（rope-first 路径）共享。PR body 指出内核 "is latency-bound with low occupancy: the baseline launches 4-warp blocks (128 threads, launch_bounds(128,16)), which run the schedulers at only ~38% achieved warp occupancy"，NCU 显示 top stall 是 long_scoreboard（warp 等待全局访存），计算管线利用率低于 35%——"so the lever is scheduling (more warps in flight to hide the load latency), not the math"。即瓶颈在调度器吞吐而非计算，属于典型 latency-bound 内核优化场景。

实现拆解

1. Profiling 定位瓶颈：在 B200（sm_100）上用 NCU 分析 fused_q_indexer_rope_hadamard_quant，确认 top stall 为 long_scoreboard（warp 等待全局访存），计算管线利用率低于 35%，判定是调度问题而非计算问题，杠杆是增加在飞 warp 数。
2. 核心改动：在 python/sglang/kernels/jit/csrc/deepseek_v4/main_norm_rope.cuh 中将 kFusedQBlockSize 从 128 改为 256。该常量为三个 Q 内核共享（fused_q_norm_rope、fused_q_indexer_rope_hadamard_quant、fp4-quant 变体），kFusedQNumWarps、共享内存尺寸、work_id 计算与 launch grid 全部派生自它，一处改动同时惠及三条路径。占用率升至约 86%，256 线程是收益拐点（寄存器保持 21/thread，384/512 线程无额外收益）。
3. 评审驱动的简化历程：初始版本包含 persistent grid-stride + 单波 grid cap + 编译期开关 -DQ_BLOCK_SIZE / -DQ_MIN_BLOCKS_PER_SM。DarkSharpness 建议先试纯 CTA 放大；作者消融显示 persistent 路径在 mid batch 中性、B≥2048 反而慢约 3%，遂删除；

随后按评审意见删除编译期开关与独立常量，最终净 diff 为一行常量 + 注释。过程中保留 lane-0-only weights_out 写（scale/weight 对 warp 内 32 lanes 统一，其余 31 次同址写是浪费）。

4. 测试与验证配套：新增 test/registered/kernels/ops/attention/test_dsv4_indexer_quant.py（189 行）：对 V4（rope-trailing + Hadamard）与 V3.2/GLM（rope-first 无 Hadamard）两条模板路径分别用 torch 参考实现校验 fp8 反量化误差与 weights_out（atol/rtol 1e-3），batch 覆盖 1-2048、position 覆盖 int32/int64，另加 strided weight 与 contiguous 等价性测试，共 19 例。作者另在 11 个 batch（1-16384）上对旧内核做 byte-exact 对照（q_fp8 0 字节差异、weights_out 0 元素差异）。CI 适配：补 __main__ 入口以满足注册式测试 runner 的 ci_register 要求，并注册 CUDA/AMD CI 分组。
5. CI 验证：JIT kernel 相关检查（unit / multigpu / b200 / AMD）全部通过，端到端 test_deepseek_v4_flash_fp4_h200.py 通过；反复失败的 DeepEP（H100/H200/B200）用例经排查为 runner 环境问题，与本次改动无关，作者在无关 PR #30281、#30677 中复现了相同错误。

关键文件：

- python/sglang/kernels/jit/csrc/deepseek_v4/main_norm_rope.cuh（模块 JIT 内核；类别 source；类型 core-logic；符号 kFusedQBlockSize, kFusedQNumWarps）：本 PR 唯一的源码改动：kFusedQBlockSize 128→256，被三个 Q 内核共享，是全部性能收益的来源；warp 数、共享内存与 launch grid 均派生自该常量。
- test/registered/kernels/ops/attention/test_dsv4_indexer_quant.py（模块 内核测试；类别 test；类型 test-coverage；符号 _skip_if_unavailable, _hadamard_matrix, _fp8_dequant_ok, test_v4_rope_hadamard_quant_matches_reference）：新增 189 行正确性测试，覆盖 V4（rope-trailing + Hadamard）与 V3.2/GLM（rope-first）两条模板路径、6 种 batch、int32/int64 positions 与 strided weight，是确保调度改动无精度回归的关键保障。

关键符号：fused_q_indexer_rope_hadamard_quant, fused_q_indexer_rope_first_quant, FusedQIndexerRopeHadamardQuantKernel, test_v4_rope_hadamard_quant_matches_reference, test_v32_rope_first_quant_matches_reference, test_v4_strided_weight_matches_contiguous, _fp8_dequant_ok

关键源码片段

python/sglang/kernels/jit/csrc/deepseek_v4/main_norm_rope.cuh

本 PR 唯一的源码改动：kFusedQBlockSize 128→256，被三个 Q 内核共享，是全部性能收益的来源；warp 数、共享内存与 launch grid 均派生自该常量。

```
// 8 warps per block：warp-per-(token, head) 工作项分发（Q 内核调度）。
// 256 线程把 fp8-quant 路径的调度器占用率从约 38% 提到约 86%，
// 用更多在飞 warp 掩盖 long_scoreboard（warp 等待全局访存）停顿。
// 数学路径（RoPE / 128 点 Hadamard / fp8 动态量化）完全不变，
// 输出与旧内核逐位一致（bitwise-identical）。
// 本 PR 的净改动就是下面这一行常量：128 -> 256。三个 Q 内核
//  （norm-rope、fp8-quant、fp4-quant）共享 kFusedQBlockSize，
```

```
// warp 数、共享内存尺寸、work_id 与 launch grid 全部派生自它，
// 因此一次改动同时让三条路径获得更高占用率。
constexpr uint32_t kFusedQBlockSize = 256;
constexpr uint32_t kFusedQNumWarps = kFusedQBlockSize / device::kWarpThreads; // 8 warps / block
```

test/registered/kernels/ops/attention/test_dsv4_indexer_quant.py

新增 189 行正确性测试，覆盖 V4（rope-trailing + Hadamard）与 V3.2/GLM（rope-first）两条模板路径、6 种 batch、int32/int64 positions 与 strided weight，是确保调度改动无精度回归的关键保障。

```
def _fp8_dequant_ok(q_fp8, ref, scale):
    """fp8-e4m3 舍入误差容限：相对误差 <= 1/16，另加底部一个 scale 步长。"""
    deq = q_fp8.float() * scale
    err = (deq - ref).abs()
    return (err <= 0.0625 * ref.abs() + scale).all()


# V4 路径：尾部 64 维 RoPE（交错）+ 128 点 Hadamard + fp8 动态量化。
# 用 torch 参考实现校验 JIT 内核输出，是调度改动无精度回归的关键保障。
@pytest.mark.parametrize("pos_dtype", [torch.int32, torch.int64])
@pytest.mark.parametrize("batch", BATCHES) # BATCHES = [1, 8, 64, 256, 512, 2048]
def test_v4_rope_hadamard_quant_matches_reference(batch, pos_dtype):
    _skip_if_unavailable()
    q = torch.randn(batch, N_HEADS, HEAD_DIM, dtype=torch.bfloat16, device="cuda")
    weight = torch.randn(batch, N_HEADS, dtype=torch.bfloat16, device="cuda")
    positions = torch.randint(0, 4096, (batch,), device="cuda", dtype=pos_dtype)

    # 调用 JIT 内核（fused_q_indexer_rope_hadamard_quant），得到 fp8 的 q 与量化 weight
    q_fp8, weights_out = fused_q_indexer_rope_hadamard_quant(
        q, weight, weight_scale, freqs_cis, positions
    )
    torch.cuda.synchronize()

    # torch 参考：尾部 64 维交错 RoPE -> 128 点 Hadamard * rsqrt(128)
    # -> 按 (token, head) 取 abs-max 做动态 fp8-e4m3 量化
    qf = q.float()
    fc = freqs_cis[positions.long()]
    cos, sin = fc.real[:, None, :], fc.imag[:, None, :]
    tail = qf[..., ROPE_DIM:]
    re, im = tail[..., 0::2], tail[..., 1::2]
    ntail = torch.stack([re * cos - im * sin, re * sin + im * cos], dim=-1).flatten(-2)
    qrot = torch.cat([qf[..., :ROPE_DIM], ntail], dim=-1)
    y = torch.matmul(qrot, _hadamard_matrix(HEAD_DIM, "cuda")) * (HEAD_DIM**-0.5)
    scale = torch.clamp(y.abs().amax(dim=-1, keepdim=True), min=1e-4) / FP8_MAX

    w_ref = weight.float() * weight_scale * scale.squeeze(-1)
    torch.testing.assert_close(weights_out.squeeze(-1), w_ref, atol=1e-3, rtol=1e-3)
    assert _fp8_dequant_ok(q_fp8, y, scale), "V4 fp8 dequant error out of tolerance"
```

```
assert torch.isfinite(q_fp8.float()).all() and torch.isfinite(weights_out).all()
```

评论区精华

评审核心围绕"如何用最小改动拿到收益"，四次交锋很有价值：

1. persistent kernel 是否必要（DarkSharpness）：开篇即质疑 "Why do we need to use persistent kernel? What about the performance of simply increase the CTA size to 256/512 threads and add __launch_bounds__similar to this impl?" 作者用消融数据回应：persistent 相对纯 CTA 放大在 B=256/512 收益为 0，B≥2048 反而慢约 3%——"once occupancy is saturated, the extra loop bookkeeping only costs"，随后删除该路径。
2. 低延迟与高吞吐差异（DarkSharpness，批准前最后提问）："what's the performance difference of block size = 128/256 ... For low latency & high throughput case, sometimes the result may be different." 作者给出 128t vs 256t 全 batch 对比：唯一回退是 B=1（约慢 4%，launch-bound），B≥8 全赢，收益单调增长至约 22%。
3. diff 收敛（DarkSharpness）："Can we clean up the diff? I guess the only diff needed is to change the default kFusedQBlockSize? ... 128 -> 256 may also bring benefit to other variants." 作者把 template 参数、独立常量与 lane-0 写优化全部折叠，净 diff 只剩一行常量，并确认三个 Q 内核共享该常量，编译产物与中间版本一致。
4. CI 噪音排查（作者主动）：DeepEP 相关 ImportError 反复失败，作者指出 deepep.py 中宽泛的 except ImportError 吞掉真实异常，并在无关 PR（#30281、#30677）复现同一错误，判定为 runner 环境问题而非本 PR 回归。

- persistent kernel 是否必要 vs 单纯放大 CTA (design): 作者删除 persistent grid-stride 与单波 grid cap，只保留 CTA 放大；PR 的净改动从包含脚手架的 190+ 行收敛到最终的一行常量变更。
- 128 与 256 block size 在低延迟 / 高吞吐场景的差异 (performance): 确认 256 为稳定最优配置，DarkSharpness 随后批准合入。
- diff 收敛为仅改 kFusedQBlockSize (design): 三个 Q 内核共享 kFusedQBlockSize，warp 数、共享内存、grid 均派生自该常量，一处改动同时生效；编译后的 launch 配置与中间版本完全一致。
- DeepEP CI 失败与 PR 无关性的排查 (other): JIT kernel 相关检查与 DSV4 端到端测试均绿；DeepEP 失败被认定为环境噪音，未阻塞合入。

风险与影响

- 风险：
  1. 低延迟场景回退：B=1 纯 launch-bound 场景慢约 4%（3104→3216 ns），grid 太小无法填满 SM，更大 CTA 帮不上忙；对延迟敏感的单 token 请求有轻微负面影响。
  2. 共享常量波及面：kFusedQBlockSize 被三个 Q 内核共享（norm-rope、fp8-quant、fp4-quant），PR 只对 fp8-quant 路径做了 byte-exact 验证，norm-rope 与 fp4-quant 路径未单独验证 256 配置的逐位一致性（虽数学路径与 block 映射关系未变，风险较低）。

3. 文档残留不一致：测试文件 docstring 与注释仍描述已删除的 grid-stride 分支语义（"after the grid-stride + occupancy scheduling optimization"），与最终实现不符，易误导后续维护者。
4. 平台差异：收益与拐点数据均来自 B200（sm_100），其他 GPU（如 H100）上最优 block size 未必是 256；常量未保留编译期覆盖开关，跨平台调优需要改源码。
5. CI 环境稳定性：DeepEP / H200 runner 环境问题在本 PR 生命周期内反复出现，合入后相关 e2e 用例可能偶发失败，需留意是否为环境噪音。 - 影响：影响范围集中在 DSA 索引器路径（DeepSeek-V4 与 V3.2/GLM 共用），对 prefill 阶段 Q 投影与量化有直接收益：大 batch（≥1024）内核耗时降低约 14%-22%，中等 batch 约 10%-15%，小 batch 基本持平或略慢。对用户完全透明：输出逐位一致、无需配置变更、无 API 变化。对团队而言，本 PR 确立了“先做最小 CTA 改动、用消融数据砍复杂度”的 kernel 优化工作流，并为 DSA 系列内核后续调度调优（K 内核、fp4 路径）提供了可复用的验证手段（torch reference 测试 + byte-exact 对照）。 - 风险标记：低延迟场景约 4% 回退，共享常量未单独验证其余内核，测试注释残留 grid-stride 描述，DeepEP CI 环境噪音

关联脉络

- PR #34421 [AMD][Perf] Fuse GatedDeltaNet QKVZBA split/reshape/cat into a single Triton kernel for Qwen3.5-architecture MoE on HIP: Qwen3.5 与 DeepSeek-V4 同属 DSA 稀疏注意力架构家族，该 PR 与本 PR 都在对 fused Q 投影 / 索引器类内核做融合与调度调优，属于同一性能优化脉络。
- PR #34642 Revert "[Kimi K3] Fuse MLA gate projection into QKV-A GEMM": Kimi K3 MLA gate 融合优化因长序列回归被回滚，提示 JIT 内核调度 / 融合类改动需要覆盖长序列与多种 batch 的验证，与本 PR 的 bitwise-identical + 全 batch 扫描验证策略形成对照。