PR #34277 完整报告

sgl-project/sglang

[DSV4] Emit TMA-aligned UE8M0 scales for FP8 einsum

合并时间：2026-08-17 13:06

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/34277

执行摘要

- 一句话：DSV4 量化内核直出 TMA 对齐 scale，decode 延迟降 1.4%
- 推荐动作：值得精读。核心设计决策是 producer-consumer 布局 co-design：让量化 kernel 直接产出消费者（DeepGEMM fp8_einsum）所需的 TMA-aligned packed 布局，省掉中间转换 kernel，这是 kernel 优化的典型思路。其次值得关注的是补零处理分配器垃圾、TensorMatcher/RuntimeCheck 对布局的严格校验，以及测试中利用 DeepGEMM 官方 layout helper 做参考、并验证 packed 与 fp32 路径 bit-exact 一致的做法。适合做算子性能优化或 GPU kernel 开发的参考样例。

功能与动机

PR body 明确说明这是对 #27926 的 follow-up：DSV4 仍输出 FP32 activation scales，所以 DeepGEMM 在每次 fp8_einsum 前都要运行 transpose_and_pack_fp32_into_ue8m0（61 次 launch，BS1 decode 每步约 114 µs）。目标是从 DSV4 quant kernel 直接产出 DeepGEMM 的 TMA-aligned packed UE8M0 int32 scale 布局，省掉中间转换 kernel，同时保证量化输出和 einsum 数值不变。

实现拆解

实现拆解（按变更顺序）：

1. 量化内核写入逻辑改造（python/sglang/kernels/jit/csrc/deepseek_v4/fp8_wo_a_group_major_quant.cuh）：
   - fp8_wo_a_group_major_quant_ue8m0_kernel 的输出 scale 指针从 float* 改为 int32_t*，新增 UE8M0_SCALES_PER_PACK = 4 常量并用 static_assert 与 sizeof(int32_t) 绑定。
   - 每个 hidden_group 的 UE8M0 指数直接按字节写入对应 int32 pack：hidden_pack_idx = hidden_group / 4、pack_byte_idx = hidden_group % 4，物理偏移按 [outer_group, packed_hidden, aligned_token] 计算，token 维连续且对齐到 4，满足 TMA 读取模式。
   - 增加补零逻辑：写入最后一个 hidden_group 时把同 pack 内剩余字节清零，避免 allocator 残留垃圾被当作 activation scale 送入 DeepGEMM。
   - FP8WoAGroupMajorQuantUE8M0Kernel::Call 同步更新 TensorMatcher 与 RuntimeCheck：shape 校验改为 packed_hidden_dim_groups 与 aligned_num_tokens

，并新增 output_s 基址 16 字节对齐检查。

2. Python 包装层调整（python/sglang/kernels/ops/attention/dsv4/fp8_wo_a.py）：
   - sglang_per_token_group_quant_fp8_dsv4_wo_a 按 packed_hidden_groups = ceil((D/128)/4) 与 aligned_num_tokens = ceil(T/4)*4 分配 torch.int32 存储张量（物理布局 [G, packed_hidden, aligned_token]）。
   - 调用 kernel 后通过两次 transpose 返回逻辑视图 [T, G, ceil((D/128)/4)]，并用 [:, :num_tokens, :] 切片隐藏 token 维 padding，对外 API 形状与语义保持不变。
3. 测试配套更新（test/registered/kernels/ops/attention/test_fp8_wo_a.py）：
   - _assert_matches_flat_reference 改用 DeepGEMM 官方布局 helper get_mn_major_tma_aligned_packed_ue8m0_tensor 生成 packed 参考，并校验 o_s 的 shape（[T, G, packed_hidden_groups]）、dtype（int32）与 stride。
   - einsum 测试重命名为 test_fp8_wo_a_einsum_uses_tma_aligned_activation_scales，新增对照组：同一组输入分别用 packed scale 与 fp32 group-major scale 调用 deep_gemm.fp8_einsum，断言输出 torch.equal 完全一致（bit-exact）。
   - 覆盖用例新增 T=1 场景，并修正空 token 维度下的预期 shape（[0, 3, 1]）。CI 注册为 base-b-kernel-unit、4-gpu-b200 runner，要求 SM 100+。

关键文件：

- python/sglang/kernels/jit/csrc/deepseek_v4/fp8_wo_a_group_major_quant.cuh（模块 量化内核；类别 source；类型 core-logic；符号 fp8_wo_a_group_major_quant_ue8m0_kernel, FP8WoAGroupMajorQuantUE8M0Kernel）：核心 CUDA kernel 修改：输出 scale 由 float32 改为 int32 packed UE8M0，物理布局变为 [G, ceil(H/4), align_up(T,4)]，并新增 pack 尾部补零逻辑，是本次性能收益的源头。
- python/sglang/kernels/ops/attention/dsv4/fp8_wo_a.py（模块 量化封装；类别 source；类型 core-logic；符号 sglang_per_token_group_quant_fp8_dsv4_wo_a）：Python 包装层：按 DeepGEMM 的 TMA-aligned 布局分配 int32 存储，并通过转置返回逻辑视图 [T, G, ceil(H/4)]，是对外 API 与底层布局之间的桥接。
- test/registered/kernels/ops/attention/test_fp8_wo_a.py（模块 算子测试；类别 test；类型 test-coverage；符号 test_fp8_wo_a_einsum_uses_tma_aligned_activation_scales, _assert_matches_flat_reference）：测试配套：用 DeepGEMM 官方布局 helper 校验 packed scale 的 shape/stride，并验证 packed 与 fp32 scale 两条 einsum 路径 bit-exact 一致，锁定数值行为。

关键符号：fp8_wo_a_group_major_quant_ue8m0_kernel, FP8WoAGroupMajorQuantUE8M0Kernel::Call, sglang_per_token_group_quant_fp8_dsv4_wo_a, _assert_matches_flat_reference, test_fp8_wo_a_einsum_uses_tma_aligned_activation_scales

关键源码片段

python/sglang/kernels/jit/csrc/deepseek_v4/fp8_wo_a_group_major_quant.cuh

核心 CUDA kernel 修改：输出 scale 由 float32 改为 int32 packed UE8M0，物理布局变为 [G, ceil(H/4), align_up(T,4)]，并新增 pack 尾部补零逻辑，是本次性能收益的源头。

```
// fp8_wo_a_group_major_quant.cuh 的关键写入逻辑：
// 输出 scale 由 float32 改为 int32，每个 int32 打包 4 个 UE8M0 指数字节，
// 物理布局为 [G, ceil((D/128)/4), align_up(T, 4)]，token 维连续且 4 对齐，
// 可直接被 DeepGEMM 的 fp8_einsum 用 TMA 读取，省去布局转换 kernel。

constexpr int UE8M0_SCALES_PER_PACK = 4;
static_assert(UE8M0_SCALES_PER_PACK == sizeof(int32_t));

// 每个 hidden_group 完成 absmax -> UE8M0 指数转换后，
// 由 lane_id == 0 的线程负责把指数字节写入对应的 int32 pack。
if (lane_id == 0) {
    const int hidden_pack_idx = hidden_group / UE8M0_SCALES_PER_PACK; // 属于第几个 int32
    const int pack_byte_idx = hidden_group % UE8M0_SCALES_PER_PACK; // pack 内第几个字节
    // 物理布局 [outer_group, packed_hidden, aligned_token]，
    // token 维连续且按 4 对齐，正好满足 TMA 的读取模式。
    const int64_t scale_word_offset =
        (static_cast<int64_t>(outer_idx) * packed_hidden_dim_groups + hidden_pack_idx) *
            aligned_num_tokens +
        token_idx;
    auto* scale_output =
        reinterpret_cast<uint8_t*>(output_s) +
        scale_word_offset * UE8M0_SCALES_PER_PACK + pack_byte_idx;
    *scale_output = static_cast<uint8_t>(scale_ue8m0);

    // DeepGEMM 按完整 int32 消费 scale。若当前 hidden_group 是最后一组，
    // 必须把 pack 内剩余字节清零，避免 allocator 的残留垃圾被当成 activation scale。
    if (hidden_group == hidden_dim_groups - 1) {
#pragma unroll
        for (int byte_idx = pack_byte_idx + 1; byte_idx < UE8M0_SCALES_PER_PACK; ++byte_idx)
        {
            scale_output[byte_idx - pack_byte_idx] = 0;
        }
    }
}
```

python/sglang/kernels/ops/attention/dsv4/fp8_wo_a.py

Python 包装层：按 DeepGEMM 的 TMA-aligned 布局分配 int32 存储，并通过转置返回逻辑视图 [T, G, ceil(H/4)]，是对外 API 与底层布局之间的桥接。

```
# fp8_wo_a.py：直接为 DeepGEMM 分配 TMA-aligned 的 packed scale 存储，
# 省去 fp8_einsum 前的 transpose_and_pack_fp32_into_ue8m0 转换 kernel。
num_tokens, num_groups, hidden = x.shape
hidden_groups = hidden // _GROUP_SIZE
packed_hidden_groups = (
    hidden_groups + _UE8M0_SCALES_PER_PACK - 1
) // _UE8M0_SCALES_PER_PACK
```

```
    aligned_num_tokens = (
        (num_tokens + _UE8M0_SCALES_PER_PACK - 1)
        // _UE8M0_SCALES_PER_PACK
        * _UE8M0_SCALES_PER_PACK
    )

    x_q = torch.empty(x.shape, device=x.device, dtype=torch.float8_e4m3fn)
    # kernel 要求物理布局 [G, packed_hidden, aligned_token] 的 int32 张量
    x_s_storage = torch.empty(
        (num_groups, packed_hidden_groups, aligned_num_tokens),
        device=x.device,
        dtype=torch.int32,
    )
    if x.numel() > 0:
        fp8_wo_a_group_major_quant_ue8m0(x, x_q, x_s_storage)

    # 两次转置得到逻辑视图 [T, G, packed_hidden]:
    # 底层 token 维连续、packed-hidden 步长为 aligned_num_tokens，即 DeepGEMM 原生布局。
    x_s = x_s_storage.transpose(-1, -2)[:, :num_tokens, :].transpose(0, 1)
    return x_q, x_s
```

评论区精华

PR 无内联 review 评论，Fridge003 直接 APPROVED。主要讨论集中在 CI 流程：b8zhong 请求 rerun test_deepseek_v4_flash_fp4_b200.py 时 dispatch 失败（422），他推测是 PR 未开放 maintainer 编辑权限所致并请作者开启；作者开启后 Fridge003 重跑 6 个 fp8/fp4 B200/H200 用例，仍有一半 dispatch 422，但最终 run-ci-extra 通过。未出现实现方案层面的技术争议。

- rerun-test 频繁 Dispatch 422 与 maintainer 编辑权限 (other): PR 状态最终 run-ci-extra 通过并合并；部分 422 由机器人 dispatch 流程问题导致，与代码本身无关。
- 整体评审：无内联技术评论直接批准 (other): 批准合并。

风险与影响

- 风险：
  1. DeepGEMM 内部布局契约耦合：output_s 的物理布局 [G, ceil(H/4), align_up(T,4)] 与 deep_gemm.utils.layout.get_mn_major_tma_aligned_packed_ue8m0_tensor 的约定强绑定，deep_gemm 版本升级时该布局若变动，DSV4 quant kernel 与测试参考都会失效，需要同步跟进。
  2. padding 字节安全：hidden 维 pack 尾部补零只在最后一个 hidden_group 写入时执行，依赖 hidden_group == hidden_dim_groups - 1 分支；token 维 padding 未初始化，但通过 wrapper 的 [:, :num_tokens, :] 切片不会暴露。若未来出现直接访问底层存储的调用方，存在读到垃圾数据的风险。
  3. 覆盖范围有限：测试要求 SM 100+（B200）且依赖 deep_gemm，CI 仅覆盖 B200 场景；H200 等 SM90 平台无此路径测试。FP8 wo_a 路径仅 DSV4 使用，其他模型不受

影响。

4. 回归风险较低：量化输出码与 flat 参考 bit-exact 一致，einsum 输出与 fp32 scale 路径 bit-exact 一致，数值行为被测试锁定。 - 影响：对用户：DeepSeek-V4 系列 decode 延迟几何平均降低 1.42%（BS16 降 2.38%、BS32 降 3.59%），吞吐提升 1.44%，且 GSM8K 精度不变。对系统：每个 BS1 decode step 省掉 61 次 kernel launch（约 114 μs），减少 GPU 调度开销。对团队：引入对 DeepGEMM 内部 scale 布局的依赖，后续 deep_gemm 升级或新增量化后端时需同步维护；测试模式（用官方 layout helper 校验 packed 布局、双路径 bit-exact 对比）可作为后续 kernel 优化的参考模板。 - 风险标记：依赖 DeepGEMM 内部布局契约，仅 B200/SM100+ 测试覆盖，padding 字节需手动清零，deep_gemm 升级可能破坏布局

关联脉络

- PR #33480 [AMD] Support prefill context parallel two batch overlap for DeepSeek V4: 同属 DeepSeek V4 模型线的性能优化工作，共享 dsv4/ 两批重叠等底层调度与量化路径，可以一起理解 DSV4 在 B200/AMD 上的性能演进。
- PR #34982 [misc] Rename shared-read boundary to shared-read ends and fix wrapper delegation: 同为 DeepSeek 系列 decode 路径的调度 /backend 修复，涉及 dsv4 相关 attention backend，属于同一功能线的收尾与稳定性工作。