PR #32395 完整报告

sgl-project/sglang

[MoE] Single-launch moe_align for tiny batches with many experts

合并时间：2026-08-08 16:08

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/32395

执行摘要

- 一句话：单 CTA Triton align 内核，让宽专家 MoE 解码每层少一次 launch
- 推荐动作：值得精读，尤其是对 Triton 内核优化和 kernel 测试感兴趣的工程师。核心看三处：pair 轴 vs expert 轴的推导与实测结论（为什么 pair 轴是唯一赢的方案）、两处有意语义偏差的契约论证（桶内稳定序、尾部不写）、以及 gate 四条件的取舍（为什么 64 桶以下留给 CUDA）。测试文件里的纯 torch oracle 与 blockwise 双重校验写法可以直接借鉴到其他 kernel 测试。

功能与动机

PR body 明确指出：CUDA 已有的 small-batch 单内核变体被限制在 num_experts <= 64（共享内存按 O(threads x experts) 增长），因此专家维度宽的模型在 bs=1 decode 时永远走通用两发射路径（align + count_and_sort）。而 bs=1 时 numel（num_tokens * topk）只有几十对，整个 decode 步骤由成百上千个小内核 back-to-back 组成，是 launch-bound 而非 compute-bound——"That is exactly the case where the launch overhead dominates the arithmetic most"，每 MoE 层每 decode 步省一次 launch 在 trace 上直接可见。

实现拆解

按 4 步拆解：

1. 新增单 CTA Triton 内核（python/sglang/kernels/ops/moe/moe_align_small_numel.py，+147行）：定义_moe_align_small_numel_kernel与launchermoe_align_small_numel。内核只在一个 CTA 内完成全部工作，核心是 pair 轴公式——用 [NP, NP] 两两比较同时算出每个 pair 的桶内稳定排名（rank）、桶人口（cnt）、填充后计数（padded_cnt），并由每桶唯一的 rank-0 代表累加出桶序独占偏移 excl 和总填充量 total。PR body 强调这不是口味问题：expert 轴（直方图 + 约 1k 桶的 cumsum）会让单 SM 关键路径工作量多约 3 倍，实测比被替换的两内核还慢。
2. runner 调用点接入 fast path（python/sglang/srt/layers/moe/moe_runner/triton_utils/moe_align_block_size.py，+32 行）：在 moe_align_block_size 分配完输出缓冲后插入四个条件的 gate——_is_cuda 且 numel <= SMALL_NUMEL_LIMIT(64) 且 num_experts + 1 > 64 且 not ignore_invalid_expert。选择在调用点显式 gate 而非 shape-driven 自动选核，是因为本框架的 kernel 选择是显式约定；ignore_invalid_expert 与 "+1 offset" 是两套过滤语义，必须留给 CUDA 路径。

3. KernelSpec 注册（python/sglang/kernels/ops/moe/__init__.py，+17 行）：按 RFC #29630 将内核注册为 op="moe.moe_align_small_numel"、backend=KernelBackend.TRITON 的 KernelSpec，能力限定 CUDA，进入 registry inventory，与既有 AOT/JIT moe.moe_align_block_size 条目并列。
4. 测试与 CI 配套（test/registered/kernels/ops/moe/test_moe_align_small_numel.py，+214 行）：注册到 base-b-kernel-unit / 1-gpu-large，用 get_ci_test_range 收窄 per-commit 参数范围。测试含四个关键用例：test_matches_reference（对纯 torch oracle 做精确比较、对 CUDA 路径做逐块 multiset 交叉校验，expert 数横跨 8/64/65/129/1024）、test_ep_filtered_ids_map_to_expert_minus_one（EP 过滤 -1 落入桶 0 且块 expert_id 为 -1）、test_runner_dispatch_boundary（numel = 63/64/65 三侧边界）、test_runner_defers_for_ignore_invalid_expert（钉死第三个 gate 条件）。

关键文件：

- python/sglang/kernels/ops/moe/moe_align_small_numel.py（模块 内核层；类别 source；类型 core-logic；符号 _moe_align_small_numel_kernel, moe_align_small_numel）：本 PR 核心：新增单 CTA Triton 内核，用 pair 轴 [NP, NP] 两两比较一次发射完成 moe_align 全部工作，覆盖 CUDA 小批量路径够不到的 num_experts > 64 corner。
- python/sglang/srt/layers/moe/moe_runner/triton_utils/moe_align_block_size.py（模块 运行器；类别 source；类型 dependency-wiring；符号 moe_align_block_size）：moe_runner 调用点，插入四条件 fast path gate，决定新内核何时接管两发射路径，是行为切换的关键位置。
- test/registered/kernels/ops/moe/test_moe_align_small_numel.py（模块 单元测试；类别 test；类型 test-coverage；符号 _reference, _alloc, _run_triton, _run_cuda）：测试覆盖完整：纯 torch oracle 精确断言、CUDA 路径逐块 multiset 交叉校验、EP 过滤语义、runner dispatch 边界（63/64/65）及 ignore_invalid_expert 回退。
- python/sglang/kernels/ops/moe/__init__.py（模块 内核注册；类别 infra；类型 configuration）：按 RFC #29630 将新内核注册为 TRITON KernelSpec（op=moe.moe_align_small_numel），纳入 registry inventory，是该内核可被发现和统一管理的基础设施改动。

关键符号：_moe_align_small_numel_kernel, moe_align_small_numel, moe_align_block_size

关键源码片段

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

本 PR 核心：新增单 CTA Triton 内核，用 pair 轴 [NP, NP] 两两比较一次发射完成 moe_align 全部工作，覆盖 CUDA 小批量路径够不到的 num_experts > 64 corner。

```
# python/sglang/kernels/ops/moe/moe_align_small_numel.py
# 单发射、单 CTA 的 moe_align：专为 numel 极小、专家数任意（可远超 64）的
# bs=1 decode corner 设计。CUDA 小批量路径的共享内存门槛是 O(threads x experts)，
# 专家多了就只能走两次发射的通用 align + count_and_sort。
SMALL_NUMEL_LIMIT = 64
```

```
@triton.jit
def _moe_align_small_numel_kernel(
    topk_ids_ptr, # [numel] 展平的 (token, slot) expert id，-1 表示 EP 过滤
    sorted_token_ids_ptr, # [max_num_tokens_padded] 输出置换
    expert_ids_ptr, # [max_num_m_blocks] 每个 block 的 expert id
    num_tokens_post_pad_ptr, # [1] 发布的有效总量
    num_experts, # E + 1：即 "+1 offset" 约定下的桶个数
    block_size,
    numel,
    NP: tl.constexpr, # power-of-2，且 >= numel
    NB: tl.constexpr, # power-of-2，且 >= 最大块数
    USE_GDC: tl.constexpr = False,
):
    # sm90+ PDL 链：本内核是 router top-k 的消费者，先等待生产者
    if USE_GDC:
        tl.extra.cuda.gdc_wait()

    offs_p = tl.arange(0, NP)
    mask_p = offs_p < numel
    ids = tl.load(topk_ids_ptr + offs_p, mask=mask_p, other=-2)
    # pad lane 给一个越界桶 num_experts，之后所有统计都会被 mask 掉
    bucket = tl.where(mask_p, (ids + 1).to(tl.int32), num_experts)

    # 关键设计：全部工作在 pair 轴（[NP, NP] 两两比较）上完成。
    # 同一桶内，rank 是比自己更早的 pair 个数，cnt 是桶人口；
    # rank == 0 的那个 pair 作为桶代表，负责推导填充计数、桶序
    # 独占偏移、总量和每块的 expert_ids。expert 轴（直方图 + cumsum）
    # 在约 1k 桶时单 SM 工作量约 3 倍，实测反而更慢。
    same = (bucket[None, :] == bucket[:, None]) & mask_p[None, :] & mask_p[:, None]
    earlier = offs_p[None, :] < offs_p[:, None]
    rank = tl.sum((same & earlier).to(tl.int32), axis=1) # 桶内稳定排名
    cnt = tl.sum(same.to(tl.int32), axis=1) # 本桶人口
    padded_cnt = ((cnt + block_size - 1) // block_size) * block_size
    is_rep = (rank == 0) & mask_p

    # 桶序独占偏移：累加所有桶 id 更小代表的 padded_cnt
    smaller_rep = (bucket[None, :] < bucket[:, None]) & is_rep[None, :]
    excl = tl.sum(smaller_rep.to(tl.int32) * padded_cnt[None, :], axis=1)

    total = tl.sum(tl.where(is_rep, padded_cnt, 0), axis=0)
    tl.store(num_tokens_post_pad_ptr, total.to(tl.int32))

    # 每个代表拥有 [excl, excl + padded_cnt) 这些块，写入的 expert id 是
    # bucket - 1：桶 0（EP 过滤的 -1）因此得到 expert_ids = -1，
    # fused_moe 的 filter_expert 会跳过这些块。
    offs_b = tl.arange(0, NB)
    block_start = offs_b * block_size
    in_range = (
        (block_start[:, None] >= excl[None, :])
```

```
        & (block_start[:, None] < (excl + padded_cnt)[None, :])
        & is_rep[None, :]
    )
    eid = tl.sum(in_range.to(tl.int32) * (bucket[None, :] - 1), axis=1)
    tl.store(expert_ids_ptr + offs_b, eid.to(tl.int32), mask=block_start < total)

    # pad 槽填 numel，再散射真实 pair index。fill 和 scatter 在不同 warp 上，
    # 没有 debug_barrier 时 fill 可能追尾覆盖同一地址的 scatter 结果。
    n_fill = (total + NP - 1) // NP
    for it in range(n_fill):
        f_offs = it * NP + offs_p
        tl.store(
            sorted_token_ids_ptr + f_offs,
            tl.full([NP], 0, tl.int32) + numel,
            mask=f_offs < total,
        )
    tl.debug_barrier()
    pos = excl + rank
    tl.store(sorted_token_ids_ptr + pos, offs_p.to(tl.int32), mask=mask_p)

    # 发布给下游 GEMM 的 PDL 依赖
    if USE_GDC:
        tl.extra.cuda.gdc_launch_dependents()
```

python/sglang/srt/layers/moe/moe_runner/triton_utils/moe_align_block_size.py

moe_runner 调用点，插入四条件 fast path gate，决定新内核何时接管两发射路径，是行为切换的关键位置。

```
# python/sglang/srt/layers/moe/moe_runner/triton_utils/moe_align_block_size.py
# CUDA 小批量单发射路径的共享内存门槛：每个线程一份直方图，
# 占 4 * (buckets + 1)^2 字节，超出 64 桶就退化到通用两发射路径
_CUDA_SMALL_BATCH_MAX_BUCKETS = 64


def moe_align_block_size(topk_ids, block_size, num_experts, ignore_invalid_expert=False):
    # … 缓冲分配（sorted_ids / expert_ids / num_tokens_post_pad / cumsum_buffer）…

    # Tiny-batch fast path（bs=1 decode）：用单 CTA Triton 发射替换
    # 通用 align + count_and_sort 两次发射。四个 gate 条件缺一不可：
    # 1) 仅 CUDA；
    # 2) numel <= 64（pair 轴张量在 NP=64 时仍放得进寄存器，
    # NP=256 时溢出到 local memory，实测约 230 us/ 发射，比被替换
    # 路径还慢，所以宁可放弃更大 batch 的收益）；
    # 3) 桶数超过 64（否则 CUDA 自己已是单发射且是 O(numel) 工作量，
    # 本内核是 O(numel^2) 两两比较，不该抢这条路）；
    # 4) 不启用 ignore_invalid_expert（它和 "+1 offset" 约定是两套语义，
    # 过滤掉的 id 处理方式不一致，必须继续走 CUDA 路径）。
    if (
```

```
        _is_cuda
        and topk_ids.numel() <= SMALL_NUMEL_LIMIT
        and num_experts + 1 > _CUDA_SMALL_BATCH_MAX_BUCKETS
        and not ignore_invalid_expert
    ):
        moe_align_small_numel(
            topk_ids,
            num_experts + 1,
            block_size,
            sorted_ids,
            expert_ids,
            num_tokens_post_pad,
        )
        return sorted_ids, expert_ids, num_tokens_post_pad

    # … 其余路径原样保留（JIT align / AOT sgl_kernel moe_align_block_size）…
```

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

测试覆盖完整：纯 torch oracle 精确断言、CUDA 路径逐块 multiset 交叉校验、EP 过滤语义、runner dispatch 边界（63/64/65）及 ignore_invalid_expert 回退。

```
# test/registered/kernels/ops/moe/test_moe_align_small_numel.py
def _reference(topk_ids, block_size, num_experts):
    # 纯 torch 的 oracle：bucket = expert + 1（EP 过滤的 -1 落到桶 0），
    # 每个桶补齐到 block_size 的倍数，块按桶序排列，pad 槽填 numel，
    # 桶内按 pair index 升序放置。oracle 与内核的桶内顺序一致，
    # 所以可以精确比较而不是 multiset 比较。
    numel = topk_ids.numel()
    bucket = (topk_ids.flatten().to(torch.int64) + 1).cpu()
    counts = torch.bincount(bucket, minlength=num_experts + 1)
    padded = ((counts + block_size - 1) // block_size) * block_size
    offsets = torch.cumsum(padded, 0) - padded
    total = int(padded.sum())

    non_empty = torch.nonzero(padded, as_tuple=True)[0]
    expert_ids = torch.repeat_interleave(
        non_empty - 1, padded[non_empty] // block_size
    ).to(torch.int32)

    sorted_ids = torch.full((total,), numel, dtype=torch.int32)
    cursor = offsets.clone()
    for pair in range(numel):
        b = int(bucket[pair])
        sorted_ids[cursor[b]] = pair
        cursor[b] += 1
    return sorted_ids, expert_ids, total


def _assert_exact(got, ref, block_size):
```

```
    # 只比较发布总量以内；尾部按设计不写，这里也不断言
    got_sorted, got_expert, got_total = got
    ref_sorted, ref_expert, ref_total = ref
    assert got_total.item() == ref_total, 'num_tokens_post_pad'
    num_blocks = ref_total // block_size
    assert torch.equal(got_expert[:num_blocks].cpu(), ref_expert), 'expert_ids'
    assert torch.equal(got_sorted[:ref_total].cpu(), ref_sorted), 'sorted_token_ids'
```

评论区精华

两条有效讨论：

- DarkSharpness 在 Issue 评论区问 "Do we have any benchmark for this kernel?"——作者在 PR body 的 Speed Tests and Profiling 一节用 H20-3e 补齐了数据（发射数 2 -> 1、align 单层约 4 us、e2e 0.7511 s -> 0.750 s），并解释了 NP=256 变体因寄存器溢出约 230 us/ 发射被否决的实测依据。
- BBuf 在测试文件第 1 行评论 "Can we clean up this file?"，作者回复 "Done."，最终测试文件从 259 行收敛到 214 行，结构更紧凑。

合并操作由 BBuf 完成（state = APPROVED）。

- 内核 benchmark 数据是否齐全 (performance): 作者在 PR body 的 Speed Tests and Profiling 一节补充 H20-3e bs=1 实测：每 MoE 层发射数 2 -> 1，align 单层约 4 us，e2e decode 0.7511 s -> 0.750 s；并解释 NP=256 变体约 230 us/ 发射因寄存器溢出被否决。
- 测试文件清理 (style): 测试文件从 259 行精简到 214 行，结构更紧凑，review 通过并合并。

风险与影响

- 风险：具体风险点：
- 静默 dispatch 变更：moe_align_block_size 是每个 MoE 层每步解码的必经调用点，四个 gate 条件任何一个写错都会静默换路径而不报错，只能靠测试兜底；test_runner_dispatch_boundary 专门钉住 63/64/65 边界，test_runner_defers_for_ignore_invalid_expert 钉住第三条件。
- 寄存器溢出性能悬崖：pair 轴张量在 NP=64 时装得进寄存器（约 4 us），NP=256 时溢出到 local memory（约 230 us，比被替换的两发射还慢约 50 倍）。硬上限 SMALL_NUMEL_LIMIT=64 是防性能回退的关键，未来若调高上限必须带性能回归验证。
- warp 间共享写无天然同步：pad fill 与真实 pair 散射在不同 warp 上执行，tl.debug_barrier() 一旦被后续重构移除，fill store 可能追尾覆盖 scatter store 到同一地址，属于隐蔽正确性隐患。
- 消费者契约假设：sorted_token_ids 发布总量之后不写，依赖 fused_moe 严格只读 num_tokens_post_pad 以内；尾部被提前读会出现未定义值，消费方改动需联动验证。
- 平台覆盖：fast path 仅 _is_cuda 生效，HIP/XPU/MUSA 回退旧路径（行为安全但无收益）；且 moe_align_small_numel 的 import 被 if _is_cuda 保护，避免非 CUDA 环境 ImportError。

- sglang-kernel 版本差异：AOT 路径在桶数大于 1023 时各 wheel 版本支持不统一，测试用 CUDA_XCHECK_MAX_EXPERTS = 1023 限制交叉校验范围，新内核本身无 expert 上限，跨版本一致性主要影响测试稳定性。
- 影响：影响范围偏窄但落在核心路径：所有专家数超过 64 的 MoE 模型（如 129/1024 专家配置）在 bs=1 decode 时每层少一次 kernel launch 和 host dispatch，实测 e2e 延迟下降约 0.1%（0.7511 s -> 0.750 s on H20-3e），收益虽小但对 decode 场景有普适性，并让 trace 更干净。对团队而言，本次建立了三个可复用资产：pair 轴单 CTA 公式、PDL 链（gdc_wait / gdc_launch_dependents）接入模式、以及 oracle + 逐块 multiset 双轨测试范式；非 CUDA 平台与 batch 较大场景完全不受影响。
- 风险标记：核心解码路径 dispatch 变更，gate 边界语义敏感，寄存器溢出防护依赖硬上限，warp 间共享写依赖 barrier，仅 CUDA 路径生效

关联脉络

- PR #33903 [Inkling] silu_and_mul: replace helion kernels with plain Triton: 同属 sglang/kernels/ops/moe 目录的算子 Triton 化与 KernelSpec 注册工作，体现 moe 算子用纯 Triton + registry 的演进方向。
- PR #33889 moe: the shared-experts-fusion decision is a per-runner value the loader installs: 同属 moe_runner 调用路径的装配与语义收紧，与本 PR 的 dispatch gate 在同一路径上叠加演进，共同推进 runner 侧决策的显式化。