PR #33903 完整报告

sgl-project/sglang

[Inkling] silu_and_mul: replace helion kernels with plain Triton

合并时间：2026-08-08 15:00

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33903

执行摘要

- 一句话：Inkling MoE 激活内核迁移纯 Triton，移除 helion 依赖
- 推荐动作：值得精读。三个看点：一是 bitwise 级内核移植方法论（复刻运算顺序、仅 store 一次 cast、用独立 harness 做 88/88 逐位对比）；二是性能工程细节（BLOCK_M==1 时 mask 省略、evict_last/evict_first 缓存提示、N-fastest 光栅化、tile/warp 扫描直至编译后 PTX 指令混合与 helion 一致、63 寄存器 0 spill）；三是用两个静态 mask 内核加小 launch 启发式替代整套 AOT autotune 配置表体系的设计决策，是消除第三方运行时 codegen 依赖的典型范例。同时注意其仅 sm_100 实测的验证边界，若团队在非 Blackwell 架构上运行 Inkling，建议补充一次基准对比。

功能与动机

PR body 开宗明义："Remove the helion dependency from the Inkling MoE activation path"。动机有三层：一是依赖收敛，将两个 helion silu_and_mul 内核移植为纯 Triton（输出逐位一致），删除 helion AOT autotune 机制与 per-arch 配置表，并移除 helion==1.4 包依赖；二是消除缺陷类别，InklingBatchDenseMLP._swiglu 注释记录 helion 内核 "can produce NaNs for small shared-expert batches"（EP+DP 配置），调查显示该报告早于仓库可达历史且无法复现，作者推断根因是 helion AOT wrapper 按 (width, dtype, size-1 pattern, int64) 键在运行时编译不同内核变体、生成代码面过大，疑似 codegen/dispatch 缺陷，移植为两个静态 mask 内核可从根本上移除该类别；三是性能与可维护性，GB300 基准显示多数形状有收益，且删除配置表后新架构接入不再需要重新 autotune。

实现拆解

1. 内核移植（python/sglang/kernels/ops/moe/inkling_moe.py）：把 _silu_and_mul_helion_interleaved_kernel 与 _silu_and_mul_helion_non_interleaved_kernel 改写为纯 Triton 的 silu_and_mul_interleaved_kernel / silu_and_mul_non_interleaved_kernel（公开命名，便于 profiler 追踪）。运算顺序完全复刻 helion：gate/up 转 fp32 → gate * sigmoid(gate) * up → 可选每行权重最后乘 → store 时一次取整 cast。外层 wrapper silu_and_mul_helion 重命名为 silu_and_mul，签名与行为不变，内部用 launch 启发式选择 tile。silu_and_mul_triton（InklingBatchDenseMLP._swiglu 使用的 persistent-grid 内核，含 sm_121 num_stages 门控）保持原样。随移植删除 AOT 专用的 silu_and_mul_key 与 silu_and_mul_inputs。

2. 调用点更新（python/sglang/srt/models/inkling_common/moe.py）：唯一调用点 activation() 的导入与调用从 silu_and_mul_helion 改为 silu_and_mul。生产路径上 routed MoE 恒传 topk_weights=None，带权重分支主要由测试覆盖。
3. 删除 helion AOT 机制（python/sglang/srt/layers/moe/moe_runner/triton_utils/helion_utils.py 与 7 个 JSON 配置表）：helion_utils.py 296 行整体删除（含 helion_aot_autotune、bind_and_compile_kernel、load_autotune_data_from_json、AOTAutotuneMode 等），其唯一导入方就是 inkling_moe.py；silu_and_mul{,interleaved}_sm{90,95,100,121}.json 共 7 个配置表一并移除。
4. 依赖移除（python/pyproject.toml、python/pyproject_other.toml）：删除 helion==1.4，代码树中不再出现 import helion。
5. 测试配套（test/registered/kernels/ops/moe/test_inkling_silu_and_mul.py）：新增 159 行、47 项测试，注册到 base-b-kernel-unit 的 1-gpu-large runner。覆盖参考数值（bf16 允许 1 ulp）、两布局逐位一致、确定性、out_dtype、奇数宽度、零行、numel > 231 的 int64 偏移路径，以及小共享专家批次回归族（M = n_shared * tokens ∈ 1..16, N = 768/1024/1536）。 配套验证：作者用独立 harness（保留被删 helion 内核与配置的副本）在 GB300 上对比 88/88 用例逐位一致，包括 numel = 231 + 16384 的大张量；test_moe_preprocess.py 17 项照常通过。代表性基准（CUDA-graph 流式、min of 3）：M=6 N=1024 从 2.69 us 降到 1.31 us（2.05x）；M=96 N=1536 从 2.47 us 降到 1.44 us（1.71x）；M=384 N=1024 带权重从 3.34 us 降到 2.15 us（1.56x）；M=49152 N=768 从 42.01 us 降到 33.39 us（1.26x）；M=49152 N=3072 从 154.97 us 降到 129.04 us（1.20x）；non-interleaved 带权重 N=768 的 2 个形状为 0.97x（慢约 3%）。

关键文件：

- python/sglang/kernels/ops/moe/inkling_moe.py（模块 激活内核；类别 source；类型 core-logic；符号 silu_and_mul_interleaved_kernel, silu_and_mul_non_interleaved_kernel, silu_and_mul, silu_and_mul_key）：本 PR 的核心：两个 helion silu_and_mul 内核在此逐位移植为纯 Triton 静态 mask 内核，并重命名 wrapper 为 silu_and_mul，同时删除 AOT 专用辅助符号。
- python/sglang/srt/layers/moe/moe_runner/triton_utils/helion_utils.py（模块 调优框架；类别 source；类型 deletion；符号 helion_aot_autotune, bind_and_compile_kernel, load_autotune_data_from_json, AOTAutotuneMode）：整文件删除（296 行）：helion AOT autotune 装饰器、JSON 配置加载与内核编译绑定机制的唯一实现，其唯一导入方就是 inkling_moe.py。
- test/registered/kernels/ops/moe/test_inkling_silu_and_mul.py（模块 内核测试；类别 test；类型 test-coverage；符号 _reference, _ulp_diff_bf16, _check, test_matches_reference）：新增 159 行、47 项测试，注册到 base-b-kernel-unit：以同序 fp32 参考验证 1 ulp 数值、布局逐位一致、确定性、int64 偏移与小共享专家批次 NaN 回归。
- python/sglang/srt/models/inkling_common/moe.py（模块 专家前向；类别 source；类型 data-contract；符号 activation）：唯一调用点：activation() 从 silu_and_mul_helion 改名为 silu_and_mul，签名与行为不变，是 4 行小改动但构成数据契约更新。

- python/sglang/srt/layers/moe/moe_runner/triton_utils/configs/silu_and_mul_interleaved_sm_100.json（模块 调优配置；类别 config；类型 deletion）：代表 7 个被删除的 per-arch 配置表之一：新内核为静态 mask 内核，不再需要 useful_configs/hash_configs 查询表，sm_90/95/121 等其余 6 个 JSON 同步删除。
- python/pyproject.toml（模块 依赖声明；类别 config；类型 configuration）：打包契约变更：删除 helion==1.4 依赖，pyproject_other.toml 同步删除，import helion 从代码树消失。

关键符号：silu_and_mul_interleaved_kernel, silu_and_mul_non_interleaved_kernel, silu_and_mul, activation, helion_aot_autotune, bind_and_compile_kernel, test_matches_reference, test_small_shared_expert_batches, test_int64_offsets

关键源码片段

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

本 PR 的核心：两个 helion silu_and_mul 内核在此逐位移植为纯 Triton 静态 mask 内核，并重命名 wrapper 为 silu_and_mul，同时删除 AOT 专用辅助符号。

```
# python/sglang/kernels/ops/moe/inkling_moe.py
# 由 helion 内核逐位移植的 silu_and_mul 激活内核（interleaved 布局）：
# 输入 gateup 形状为 [M, 2N]，行内按 [g0, u0, g1, u1, ...] 交错存放，
# 计算 down_inp[i] = silu(gate[i]) * up[i]，可选再乘每行权重 topk_weights[i]。
# 移植要点：
# 1) 运算顺序与 helion 完全一致（先转 fp32，乘法后仅 store 时一次取整 cast），
# 因此 bf16 输出逐位一致；
# 2) 每个 load/store 都带显式 mask，不再依赖 helion 的运行时 codegen 与
# per-arch 配置表，从根本上消除 " 小批量共享专家产出 NaN" 的缺陷类别。

@triton.jit(do_not_specialize=["M"])
def silu_and_mul_interleaved_kernel(
    gateup_out_ptr, # [M, 2N] 输入，行内 [g0, u0, g1, u1, ...]
    topk_weights_ptr, # [M] 每行权重，仅在 HAS_TOPK_WEIGHTS=True 时读取
    down_inp_ptr, # [M, N] 输出
    M,
    stride_gm,
    N: tl.constexpr,
    HAS_TOPK_WEIGHTS: tl.constexpr,
    INT64_INDEX: tl.constexpr,
    BLOCK_M: tl.constexpr,
    BLOCK_N: tl.constexpr,
):
    # 扁平 grid，N 方向 tile 最快：相邻 CTA 读取同一行的连续片段，
    # 实测比 M 方向优先的光栅化更快。
    pid = tl.program_id(0)
    if INT64_INDEX:
        pid = pid.to(tl.int64) # numel >= 2**31 时改用 int64 寻址
    NUM_BLOCKS_N: tl.constexpr = tl.cdiv(N, BLOCK_N)
    pid_m = pid // NUM_BLOCKS_N
```

```
pid_n = pid % NUM_BLOCKS_N

offs_m = pid_m * BLOCK_M + tl.arange(0, BLOCK_M)
offs_n = pid_n * BLOCK_N + tl.arange(0, BLOCK_N)
mask_n = offs_n < N
offs_2n = pid_n * (2 * BLOCK_N) + tl.arange(0, 2 * BLOCK_N)
mask_2n = offs_2n < 2 * N
if BLOCK_M == 1:
    # grid 恰好覆盖 M，省略行 mask 在带宽受限形状上有可测收益
    mask_mn = mask_n[None, :]
    mask_m2n = mask_2n[None, :]
else:
    mask_m = offs_m < M
    mask_mn = mask_m[:, None] & mask_n[None, :]
    mask_m2n = mask_m[:, None] & mask_2n[None, :]

# 按 [g, u] 对连续加载整段数据，用 tl.split 在寄存器内去交错；
# 逐元素 stride gather 会让有效带宽减半，必须避免。
gateup = tl.load(
    gateup_out_ptr + offs_m[:, None] * stride_gm + offs_2n[None, :],
    mask=mask_m2n,
    other=0.0,
)
gate, up = tl.split(tl.reshape(gateup, (BLOCK_M, BLOCK_N, 2)))
gate = gate.to(tl.float32)
up = up.to(tl.float32)

# 运算顺序与 helion 相同：全流程在 fp32 中计算
down_inp = gate * tl.sigmoid(gate) * up
if HAS_TOPK_WEIGHTS:
    if BLOCK_M == 1:
        weight_scale = tl.load(
            topk_weights_ptr + offs_m, eviction_policy="evict_last"
        ).to(tl.float32)
    else:
        weight_scale = tl.load(
            topk_weights_ptr + offs_m,
            mask=offs_m < M,
            eviction_policy="evict_last",
        ).to(tl.float32)
    down_inp = down_inp * weight_scale[:, None]

# 仅在此处做一次 fp32 -> 输出 dtype 的取整 cast，与 helion 对齐
offs_mn = offs_m[:, None] * N + offs_n[None, :]
tl.store(
    down_inp_ptr + offs_mn,
    down_inp.to(down_inp_ptr.dtype.element_ty),
    mask=mask_mn,
)
```

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

新增 159 行、47 项测试，注册到 base-b-kernel-unit：以同序 fp32 参考验证 1 ulp 数值、布局逐位一致、确定性、int64 偏移与小共享专家批次 NaN 回归。

```
# test/registered/kernels/ops/moe/test_inkling_silu_and_mul.py
# 参考实现刻意采用与内核一致的运算顺序（整体转 fp32、weight 最后乘、
# 末尾一次 cast），因此 bf16 输出与参考只差 tl.sigmoid / torch.sigmoid
# 的最后一个 fp32 ulp，容差收紧到 1 个 bf16 ulp。

def _reference(gateup, weights, out_dtype, interleaved):
    xf = gateup.float()
    if interleaved:
        gate, up = xf[:, 0::2], xf[:, 1::2]
    else:
        n = gateup.shape[1] // 2
        gate, up = xf[:, :n], xf[:, n:]
    out = gate * torch.sigmoid(gate) * up
    if weights is not None:
        out = out * weights.float()[:, None] # 每行权重最后乘
    return out.to(out_dtype or gateup.dtype) # 末尾一次取整 cast


def _ulp_diff_bf16(a, b):
    # bf16 逐位比较：剥离符号位后按 int16 取差，得到 ulp 距离
    mask = (1 << 15) - 1
    ai = a.contiguous().view(torch.int16).to(torch.int64)
    bi = b.contiguous().view(torch.int16).to(torch.int64)
    ai = torch.where(ai < 0, -(ai & mask), ai)
    bi = torch.where(bi < 0, -(bi & mask), bi)
    return (ai - bi).abs()


# 回归旧 helion 内核在 EP+DP 配置下小共享专家批次的 NaN 报告：
# m = n_shared_experts * tokens_on_dp_rank，n 为每 rank 共享专家宽度；
# 要求输出有限且正确——这是本 PR 顺带修复的行为变更。
@pytest.mark.parametrize("m", [1, 2, 4, 8, 16])
@pytest.mark.parametrize("n", [768, 1024, 1536])
def test_small_shared_expert_batches(m, n):
    out = _check(m, n, True, True)
    assert out.isfinite().all()
```

评论区精华

本 PR 没有任何代码级 review 评论，唯一 review 是维护者 ispobock 的 APPROVED（正文为空）。仅有的 2 条 issue 评论都是 CI 命令：sshleifer 的 /tag-and-rerun-ci 与 ispobock 的 /rerun-failed-ci；提交历史中也有两次重跑 CI 的记录（空提交与加 run-ci 标签），最终 PR Test 与 PR Test Extra 均通过。实质性的技术论证全部由 PR body 承载，核心声明包括："The report predates all reachable history. The failure does not reproduce with the

pinned helion and the checked-in configs on GB300." 以及 "The suspected failure classis a codegen or dispatch defect in the helion version used at report time… the AOTwrapper compiles a distinct kernel variant per (width, dtype, size-1 pattern, int64) keyat runtime, which is a large generated-code surface." 结论是与其排查无法复现的第三方编译器问题，不如把整个运行时按 key 生成代码的机制移除。

- 无代码级 review 评论，论证由 PR body 承载 (other): 技术论证全部由 PR body 承载：88/88 逐位证据、基准表、NaN 调查报告；无未解决疑虑。
- CI 重跑与合并前状态 (testing): 最终 CI 全绿后由 ispobock 合并，无失败项遗留。

风险与影响

- 风险：覆盖验证局限：所有 bitwise 与性能验证仅在 GB300（sm_100）完成，sm_90/sm_95/sm_121 的配置表直接删除且未在对应架构复测；新内核无 arch 特化、所有访问带 mask，正确性风险低，但非 Blackwell 架构上的性能未知，存在隐性回归可能。性能残余：non-interleaved + 每行权重 + N=768 + M≥6144 的两个形状比 helion 慢约 3%（编译后 PTX 指令混合与 helion 一致，残余归因于 allocation 地址敏感），该形状不在生产路径（MoE activation() 恒传 topk_weights=None），影响可控。行为变更：凡旧 helion 栈曾产出 NaN 的输入，新内核会输出正确有限值，是有意 bugfix，但属于语义变化。打包契约：pyproject 删除 helion==1.4 影响所有安装路径，若代码树残留未发现的 import helion 会在运行时直接 ImportError，PR 声明已清理干净但只在 GB300 路径验证。测试环境依赖：test_int64_offsets 需要 ≥ 16 GB 显存否则 skip，CI 实际覆盖取决于 runner 配置。
- 影响：用户 / 部署层面：安装与运行 sglang 不再需要第三方 helion 内核编译器依赖，简化环境；Inkling 系模型（如 d42/d66）的 MoE 激活路径在 GB300 上 interleaved 布局实测提速 1.04x-2.05x，大 batch（49152 行）约 1.2x-1.26x，decode 小批量在 N=1024/1536/2048 上收益最大（1.4x-2.05x），non-interleaved 基本持平（0.97x-1.63x）。系统层面：净删约 700 行代码（含 296 行 autotune 框架与 7 个 JSON 配置表），消除启动期配置表查找、运行时 per-key codegen、hash miss 警告等路径，降低维护与分发成本。团队层面：47 项新测试为后续内核修改提供安全网，并固化了小共享专家批次、布局逐位一致性、int64 大张量等边界；该 PR 是仓库 helion 内核逐个迁移到纯 Triton 路线的样板。
- 风险标记：仅 sm_100 实测其余架构未复测，打包依赖移除契约变更，NaN 行为变更有意修复，非交错布局小 N 慢约 3%, int64 偏移测试依赖大显存 runner

关联脉络

- PR #31681 Initial Inkling import: PR body 明确引用：helion NaN 注释随 #31681 的初始 Inkling 导入进入仓库，本次移植正是为了移除该缺陷类别。
- PR #33417 Fix deterministic inference for Inkling: 同为 Inkling 内核确定性改进：前者修复 batch 形状变化时 logprob 不一致，本 PR 以逐位一致为验收标准，二者共同构成 Inkling 内核数值质量基线。
- PR #34009 Add the 8-gpu Inkling consistency test: 同在 GB300（sm_100）上验证 Inkling 内核一致性，反映仓库对 Inkling 数值一致性与 Blackwell 适配的持续投入。