PR #36502 完整报告

sgl-project/sglang

[diffusion] fuse Helios paired transposed RoPE

合并时间：2026-08-28 08:57

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36502

执行摘要

- 一句话：Helios 自注意力 QK RoPE 融合为 JIT CUDA kernel，单对 QK 提速约 11.58 倍。
- 推荐动作：值得精读。该 PR 是“eager 高频算子 → JIT CUDA kernel + custom op + 严格数值契约”的典型范本，尤其适合关注 diffusion 推理性能、JIT kernel 注册机制（register_custom_op + cache_once + load_jit）以及 torch.compile fullgraph 兼容性的读者。可重点学习：1）如何在 can_use_* 守卫中同时满足 eager 安全与 Dynamo traceable；2）如何用 torch.equal + 参考实现锁定 bit-exact 语义；3）对 process-nondeterministic 模型如何组织正确性论证（算子级 bit-exact + 统计性 SSIM 对比）。

功能与动机

PR body 明确指出：Helios 在每个自注意力块中都要对已经归一化的 Q 和 K 应用相同的转置 RoPE 链，在生产的 [8640, 40, 128] BF16 形状下，eager reshape/chunk/mul/add/stack 路径每对 Q/K 耗时约 1.62 ms，并在短去噪 profile 中产生数千次 kernel 启动。作者旨在用轻量级 JIT CUDA kernel 替换这段高频 eager 路径，同时要求精度上严格保持 eager 的 FP32 中间舍入边界，保证数值一致性。

实现拆解

实现分为五步：

1. 新增 JIT CUDA kernel（C++ 端）：在 python/sglang/kernels/jit/csrc/diffusion/helios_qk_rope.cuh 中新增 helios_qk_rope_kernel 模板与 HeliosQKRoPEKernel::run 入口。kernel 中一个线程处理一个相邻 rotary pair，通过 __fmul_rn / __fadd_rn / __fsub_rn 显式保留 FP32 round-to-nearest 的乘加 / 减中间舍入边界，再一次性 cast 回 FP16/BF16，最后原地写回 Q 与 K。
2. 新增 Python JIT 封装与守卫函数：在 python/sglang/kernels/ops/diffusion/rope/helios_qk_rope_jit.py 中实现 _jit_helios_qk_rope_module（按 dtype 缓存 JIT module）、fused_inplace_helios_qk_rope（用 @register_custom_op(mutates_args=["q", "k"]) 注册为可被 torch.compile 捕获的 custom op）和 can_use_helios_qk_rope（逐条校验 CUDA 设备、FP16/BF16 dtype、频率张量为 FP32、形状 freqs.shape == (*q.shape[:2], 2 * q.shape[-1])、连续布局、偶数头维与偶数 storage offset 对齐；在 torch.compiler.is_compiling() 时跳过 pointer/storage-offset 查询以保证 Dynamo 可 trace）。

3. 注册扩散 kernel 后端入口：在 python/sglang/kernels/ops/diffusion/__init__.py 中新增 diffusion.helios_qk_rope 的 KernelBackend.JIT 注册项，并将 can_use_helios_qk_rope / fused_inplace_helios_qk_rope 加入懒加载 export 映射，保证公开导入表面完整。
4. 接入模型前向路径：在 python/sglang/multimodal_gen/runtime/models/dits/helios.py 的 HeliosSelfAttention 中新增 _apply_rotary_qk 方法：仅当 not self.tp_rmsnorm 且 can_use_helios_qk_rope 通过时调用 fused_inplace_helios_qk_rope 并直接返回原 Q/K 张量；否则回退到原来的 apply_rotary_emb_transposed。forward 中原先的两行 RoPE 调用被替换为 q, k = self._apply_rotary_qk(q, k, rotary_emb)。模型归一化逻辑不变，TP RMSNorm 场景保持 eager 路径。
5. 测试、benchmark 与文档配套：新增单元测试 test/registered/kernels/ops/diffusion/test_helios_qk_rope.py（覆盖 FP16/BF16、head dim 64/128/256、生产形状 [8640, 40, 128]、torch.equal 逐位对比 eager 参考、原地指针保持、runtime guards、TP fallback、torch.compile(fullgraph=True) 下 custom op 与模型 dispatch，以及异常频率形状拒绝）；新增 benchmark test/registered/kernels/benchmark/diffusion/bench_helios_qk_rope.py（eager vs JIT 对比，接入 base-b-kernel-benchmark CI stage）；在 python/sglang/kernels/ops/diffusion/README.md 与 python/sglang/multimodal_gen/.claude/skills/sglang-diffusion-benchmark-profile/existing-fast-paths.md 中补充 fast-path 契约文档。

关键文件：

- python/sglang/kernels/ops/diffusion/rope/helios_qk_rope_jit.py（模块 扩散算子；类别 infra；类型 infrastructure；符号 _jit_helios_qk_rope_module, fused_inplace_helios_qk_rope, can_use_helios_qk_rope）：本 PR 核心 Python 封装：定义 JIT module 缓存、融合算子与能力守卫。can_use_helios_qk_rope 是控制 eager 回退与编译期行为的关键契约，fused_inplace_helios_qk_rope 通过 register_custom_op 支持 torch.compile fullgraph 捕获。
- python/sglang/kernels/jit/csrc/diffusion/helios_qk_rope.cuh（模块 JIT 内核；类别 other；类型 dependency-wiring；符号 helios_qk_rope_kernel, HeliosQKRoPEKernel）：实际执行融合的 CUDA kernel。核心是“一个线程处理一个相邻 rotary pair”，用 __fmul_rn/__fadd_rn/__fsub_rn 显式保持与 eager 路径一致的 FP32 舍入边界，是 bit-exact 语义的物理保证。
- python/sglang/multimodal_gen/runtime/models/dits/helios.py（模块 模型前向；类别 source；类型 data-contract；符号 _apply_rotary_qk）：模型接入点：HeliosSelfAttention._apply_rotary_qk 决定是否走融合 kernel，是性能路径与 eager 回退的分界。TP RMSNorm 与所有不兼容输入都在这里被安全导回原实现。
- test/registered/kernels/ops/diffusion/test_helios_qk_rope.py（模块 单元测试；类别 test；类型 test-coverage；符号 _reference, test_helios_qk_rope_matches_eager_transposed_path, test_helios_qk_rope_runtime_guards, test_helios_attention_dispatch_and_tp_fallback）：最完整的正确性护栏：用 _reference 实现 eager 路径，以 torch.equal 锁定 bit-exact 输出，覆盖生产形状、runtime guards、TP fallback、fullgraph 编译与异常频率形状，是数值一致性的主要证据来源。

- test/registered/kernels/benchmark/diffusion/bench_helios_qk_rope.py（模块 性能基准；类别 test；类型 test-coverage；符号 _split, benchmark）：生产形状性能证据：*eager* _split 与 JIT 融合算子在相同输入上对比，直接产出 PR 声称的 11.58 倍提速数据，并接入 CI benchmark stage 防止回归。
- python/sglang/kernels/ops/diffusion/__init__.py（模块 算子注册表；类别 infra；类型 infrastructure）：kernel 注册入口：把新 JIT kernel 加入 KernelBackend.JIT 注册表与懒加载 *export* 映射，决定 from sglang.kernels.ops.diffusion import fused_inplace_helios_qk_rope 的公开导入契约。
- python/sglang/multimodal_gen/.claude/skills/sglang-diffusion-benchmark-profile/existing-fast-paths.md（模块 开发文档；类别 docs；类型 documentation）：文档配套：向 profiling skill 记录 fast-path 契约，帮助后续开发者理解何时可用、何时回退，属于可维护性投入。
- python/sglang/kernels/ops/diffusion/README.md（模块 算子文档；类别 docs；类型 documentation）：扩散 kernel 目录 README 的契约登记，便于新 kernel 接入时保持一致文档风格。

关键符号：helios_qk_rope_kernel, HeliosQKRoPEKernel::run, _jit_helios_qk_rope_module, fused_inplace_helios_qk_rope, can_use_helios_qk_rope, _apply_rotary_qk, _reference, benchmark

关键源码片段

python/sglang/kernels/ops/diffusion/rope/helios_qk_rope_jit.py

本 PR 核心 Python 封装：定义 JIT module 缓存、融合算子与能力守卫。can_use_helios_qk_rope 是控制 *eager* 回退与编译期行为的关键契约，fused_inplace_helios_qk_rope 通过 register_custom_op 支持 torch.compile fullgraph 捕获。

```
"""Bit-exact paired RoPE for Helios' transposed frequency layout."""

from __future__ import annotations

from typing import TYPE_CHECKING

import torch

from sglang.kernels.jit.utils import cache_once, load_jit, make_cpp_args
from sglang.srt.utils.custom_op import register_custom_op

if TYPE_CHECKING:
    from tvm_ffi.module import Module


@cache_once
def _jit_helios_qk_rope_module(dtype: torch.dtype) -> Module:
    # 仅支持 FP16 / BF16，其他 dtype 直接拒绝，避免生成无意义 kernel
    if dtype not in (torch.float16, torch.bfloat16):
```

```
        raise RuntimeError(
            f"Unsupported Helios QK RoPE dtype {dtype}; expected float16 or bfloat16"
        )
    args = make_cpp_args(dtype)
    # 按 dtype 缓存 JIT 编译产物；cuda_wrappers 指向模板实例化后的 run 入口
    return load_jit(
        "helios_qk_rope",
        *args,
        cuda_files=["diffusion/helios_qk_rope.cuh"],
        cuda_wrappers=[("helios_qk_rope", f"HeliosQKRoPEKernel<{args}>::run")],
    )


# 注册为 mutating custom op：torch.compile(fullgraph=True) 也能直接捕获
@register_custom_op(mutates_args=["q", "k"])
def fused_inplace_helios_qk_rope(
    q: torch.Tensor,
    k: torch.Tensor,
    freqs: torch.Tensor,
) -> None:
    """Apply Helios' transposed RoPE to contiguous normalized Q/K in place."""
    module = _jit_helios_qk_rope_module(q.dtype)
    module.helios_qk_rope(q, k, freqs)


def can_use_helios_qk_rope(
    q: torch.Tensor,
    k: torch.Tensor,
    freqs: torch.Tensor,
) -> bool:
    """Return whether tensors match the native Helios paired-RoPE contract."""
    if q.dim() != 4 or freqs.dim() != 3:
        return False
    # Dynamo 无法 trace pointer 或 storage-offset 查询；编译路径的 Q/K 来自
    # linear 输出天然对齐，因此编译期跳过对齐检查，eager 调用者保留守卫
    pair_aligned = True
    if not torch.compiler.is_compiling():
        pair_aligned = q.storage_offset() % 2 == 0 and k.storage_offset() % 2 == 0
    return (
        q.is_cuda
        and k.is_cuda
        and freqs.is_cuda
        and q.dtype in (torch.float16, torch.bfloat16)
        and k.dtype == q.dtype
        and freqs.dtype is torch.float32
        and q.device == k.device == freqs.device
        and k.shape == q.shape
        and all(size > 0 for size in q.shape)
        and freqs.shape == (*q.shape[:2], 2 * q.shape[-1])
```

```
        and q.shape[-1] % 2 == 0
        and q.is_contiguous()
        and k.is_contiguous()
        and freqs.is_contiguous()
        and pair_aligned
    )


__all__ = ["can_use_helios_qk_rope", "fused_inplace_helios_qk_rope"]
```

python/sglang/kernels/jit/csrc/diffusion/helios_qk_rope.cuh

实际执行融合的 CUDA kernel。核心是"一个线程处理一个相邻 rotary pair"，用 __fmul_rn/__fadd_rn/__fsub_rn 显式保持与 *eager* 路径一致的 FP32 舍入边界，是 bit-exact 语义的物理保证。

```
#include <sgl_kernel/tensor.h>
#include <sgl_kernel/utils.h>
#include <sgl_kernel/type.cuh>
#include <sgl_kernel/utils.cuh>
#include <tvm/ffi/container/tensor.h>

#include <cstdint>
#include <limits>
#include <type_traits>

namespace sglang {

/**
 * 应用 Helios 转置 RoPE 到归一化后的 Q/K（原地）。
 * 一个线程负责一个相邻 rotary pair；显式的 round-to-nearest 乘法和
 * 加 / 减操作保留 eager 路径分离的 FP32 中间结果，再缩回 fp16/bf16。
 */
template <typename T>
__global__ void helios_qk_rope_kernel(
    T* __restrict__ q,
    T* __restrict__ k,
    const float* __restrict__ freqs,
    uint32_t num_pairs,
    uint32_t pairs_per_head,
    uint32_t num_heads,
    uint32_t freq_stride) {
  static_assert(std::is_same_v<T, fp16_t> || std::is_same_v<T, bf16_t>);
  using Packed = packed_t<T>;

  auto* q_pairs = reinterpret_cast<Packed*>(q);
  auto* k_pairs = reinterpret_cast<Packed*>(k);
  const uint32_t stride = blockDim.x * gridDim.x;

  // 每个线程独立处理一个 pair，通过 grid-stride 循环覆盖全部 pair
```

```
    for (uint32_t pair_index = blockIdx.x * blockDim.x + threadIdx.x;
         pair_index < num_pairs;
         pair_index += stride) {
      const uint32_t pair_in_head = pair_index % pairs_per_head;
      const uint32_t token_head = pair_index / pairs_per_head;
      const uint32_t token_index = token_head / num_heads;
      const uint32_t head_dim = pairs_per_head * 2;
      const uint32_t freq_base = token_index * freq_stride;

      // Helios 转置布局：cos 取自前一半，sin 取自后一半，且交叉索引
      const float cos = freqs[freq_base + pair_in_head * 2];
      const float sin = freqs[freq_base + head_dim + pair_in_head * 2 + 1];

      const auto q_value = device::cast<fp32x2_t, Packed>(q_pairs[pair_index]);
      const auto k_value = device::cast<fp32x2_t, Packed>(k_pairs[pair_index]);

      // 显式 __fmul_rn / __fadd_rn / __fsub_rn：保证与 eager 的 FP32 中间
      // 舍入边界完全一致，避免编译器重排导致数值漂移
      const float q_even = __fsub_rn(__fmul_rn(q_value.x, cos), __fmul_rn(q_value.y, sin));
      const float q_odd = __fadd_rn(__fmul_rn(q_value.x, sin), __fmul_rn(q_value.y, cos));
      const float k_even = __fsub_rn(__fmul_rn(k_value.x, cos), __fmul_rn(k_value.y, sin));
      const float k_odd = __fadd_rn(__fmul_rn(k_value.x, sin), __fmul_rn(k_value.y, cos));

      q_pairs[pair_index] = device::cast<Packed, fp32x2_t>(make_float2(q_even, q_odd));
      k_pairs[pair_index] = device::cast<Packed, fp32x2_t>(make_float2(k_even, k_odd));
    }
  }

  /** 校验形状并 launch 配对 Helios Q/K RoPE kernel。 */
  template <typename DType>
  struct HeliosQKRoPEKernel {
    static void run(const tvm::ffi::TensorView q, /*...*/) {
      // 运行时校验 q/k/freqs 的维度与连续性后，按 num_pairs 计算 grid/block，
      // 实例化 helios_qk_rope_kernel<DType> 并同步 launch
    }
  };

  } // namespace sglang
```

test/registered/kernels/ops/diffusion/test_helios_qk_rope.py

最完整的正确性护栏：用 _reference 实现 eager 路径，以 torch.equal 锁定 bit-exact 输出，覆盖生产形状、runtime guards、TP fallback、fullgraph 编译与异常频率形状，是数值一致性的主要证据来源。

```
  # 参考实现：严格复刻 eager 的转置 RoPE 计算顺序，作为 bit-exact 对比基准
  def _reference(value: torch.Tensor, freqs: torch.Tensor) -> torch.Tensor:
      x_1, x_2 = value.unflatten(-1, (-1, 2)).unbind(-1)
      cos, sin = freqs.unsqueeze(-2).chunk(2, dim=-1)
      out = torch.empty_like(value)
```

```
    # Helios 转置布局：偶数位用 cos 的偶下标，奇数位用 sin 的奇下标
    out[..., 0::2] = x_1 * cos[..., 0::2] - x_2 * sin[..., 1::2]
    out[..., 1::2] = x_1 * sin[..., 1::2] + x_2 * cos[..., 0::2]
    return out.type_as(value)


@pytest.mark.parametrize("dtype", [torch.float16, torch.bfloat16])
@pytest.mark.parametrize(
    "tokens,heads,head_dim",
    [
        (1, 1, 64),
        (17, 8, 128),
        (129, 4, 256),
        (8640, 40, 128), # 生产形状：Helios 实际推理时的 Q/K 尺寸
    ],
)
def test_helios_qk_rope_matches_eager_transposed_path(
    dtype: torch.dtype,
    tokens: int,
    heads: int,
    head_dim: int,
) -> None:
    generator = torch.Generator(device="cuda").manual_seed(20260826)
    q = torch.randn(tokens, heads, head_dim, device="cuda", dtype=dtype, generator=generator)
    k = torch.randn_like(q)
    freqs = torch.randn(tokens, 2 * head_dim, device="cuda", dtype=torch.float32, generator=
    generator)

    q_ref, k_ref = _reference(q, freqs), _reference(k, freqs)
    q_out, k_out = q.clone(), k.clone()
    q_ptr, k_ptr = q_out.data_ptr(), k_out.data_ptr()

    fused_inplace_helios_qk_rope(q_out, k_out, freqs)
    torch.cuda.synchronize()

    # 必须原地修改且逐位等于 eager 参考实现
    assert q_out.data_ptr() == q_ptr
    assert k_out.data_ptr() == k_ptr
    assert torch.equal(q_out, q_ref)
    assert torch.equal(k_out, k_ref)
```

评论区精华

该 PR 无 review 评论与 review 线程（comments_count 为 0，review_comments_count 为 0）。PR body 中的关键设计自述包括：1）显式 FP32 round-to-nearest 是为了“preserve the eager multiply and add/subtract boundaries before casting back to FP16/BF16”；2）编译期通过 torch.compiler.is_compiling() 跳过 storage-offset 检查，因为“Dynamo cannot trace pointer or storage-offset queries. Compiled Helios Q/K come directly from aligned

linear outputs; eager callers retain the guard”；3）Helios-Mid 与 Helios-Distilled 被明确说明为 process-nondeterministic，因此不声称字节级输出一致，正确性证据由生产形状 bit-exact 算子测试与 within-path/cross-path SSIM 矩阵提供。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 数值一致性风险：kernel 要求显式对齐 eager 的 FP32 舍入边界（__fmul_rn/__fadd_rn/__fsub_rn），若未来 eager 路径或频率生成格式变化，可能破坏 bit-exact 契约；现有测试用 torch.equal 锁定了 FP16/BF16 与三种 head dim。
  2. 守卫条件遗漏风险：can_use_helios_qk_rope 依赖一组形状 / 布局 / 对齐条件，尤其要求 freqs.shape == (*q.shape[:2], 2 * q.shape[-1]) 与偶数 storage offset；任何未覆盖的合法 eager 输入若被误判为可用，可能产生错误结果。当前测试覆盖了奇数 offset、非连续视图、空序列和错误频率形状等负例。
  3. 编译期行为差异：torch.compiler.is_compiling() 时跳过对齐检查，若编译后的输入实际未对齐（如来自非常规算子输出），存在静默错误的可能；PR 依赖“编译路径来自 linear 输出天然对齐”的假设，属于隐含契约。
  4. TP 路径覆盖风险：TP RMSNorm 场景强制走 eager 路径，多卡 TP 用户不会获得提速，但正确性不受影响。
  5. kernel 仅支持 CUDA：非 CUDA 后端（XPU/NPU/AMD 部分场景）不含此 JIT 路径，can_use_helios_qk_rope 会因 is_cuda 检查返回 False，行为安全但无性能收益。 - 影响：影响范围集中在 Helios 系列视频生成模型（Helios-Base/Mid/Distilled）在 CUDA 单卡、非 TP RMSNorm 场景下的自注意力前向路径。生产形状 microbenchmark 显示单对 QK 从 416.6 us（2160 tokens）/1617.3 us（8640 tokens）降到 39.6 us / 139.7 us，约 11.58 倍；Helios-Mid 去噪阶段提速约 1.09 倍、端到端约 1.09 倍（51.135 s → 46.838 s），Helios-Base 端到端约 1.08 倍（82.136 s → 76.095 s）。对用户而言是纯性能正向且数值等价（Helios-Base 输出文件 SHA256 一致），对团队而言新增了一个 JIT diffusion kernel 的注册与维护入口，并沉淀了 bit-exact kernel 的测试范式。 - 风险标记：bit-exact 数值契约依赖 FP32 舍入边界，can_use 守卫存在隐含对齐假设，仅 CUDA 路径受益，TP RMSNorm 场景无收益

关联脉络

- PR #36658 [multimodal_gen] fix: make tail_attn_meta CUDA-graph capturable: 同为 multimodal_gen 运行时与 CUDA graph / JIT 兼容性修复，反映该目录下 kernel 与编译捕获是持续关注点。
- PR #36726 [Diffusion] Fix the five unit tests failing on main: 涉及 multimodal_gen 下 diffusion 模型（含 dits 组件）的测试稳定性，与本 PR 的 diffusion kernel 测试同属一条 CI 质量线。
- PR #34747 [Cosmos3] Add cosmos3 transfer capability: 同为 multimodal_gen diffusion 模型（dits）新增能力并配套测试，说明该目录模型持续演进，kernel 优化会跟随

模型接入。