PR #29708 完整报告

sgl-project/sglang

[KDA-Pilot] Add LTX2 QKNorm split-RoPE CUDA fast path

合并时间：2026-07-01 14:42

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/29708

执行摘要

- 一句话：融合 LTX2 QKNorm+split-RoPE CUDA 内核，B200 端到端加速 ~9%
- 推荐动作：值得精读。重点关注：① @register_custom_op 与 fake_impl 结合 torch.compile 的技法；② 条件降级与全局禁用模式的设计；③ CUDA kernel 内 warp 级 rstd reduce 的写法。

功能与动机

PR body 指出热点模式为 `q = apply_split_rotary_emb(q_norm(q), (q_cos, q_sin)).to(torch.bfloat16)`，原始实现分两步执行 RMSNorm 和 split-RoPE，产生中间张量开销。融合内核保持 bitwise 相等的同时避免分离的计算与存取开销。

实现拆解

1. CUDA 融合内核（ltx2_qknorm_split_rope.cuh）：在 B200 (SM≥100) BF16 输入上实现了单 kernel 内的 RMSNorm (warp-level rstd reduce) + split-RoPE，支持 head_dim=64/128、非连续 cos/sin 步长。
2. Python 绑定与自定义 op（ltx2_qknorm_split_rope.py）：利用 @register_custom_op 装饰器注册为 PyTorch 自定义算子，提供 fake_impl 以便 torch.compile 能正确处理；暴露 can_use_ltx2_qknorm_split_rope_cuda 前置检查函数。
3. 模型集成（ltx_2.py）：在 LTX2Attention.forward 中优先尝试 _ltx2_try_fused_qknorm_split_rope，满足 SM100+、nn.RMSNorm、4D split-RoPE 等条件时调用 CUDA 路径；失败后全局禁用并回退到原 eager 路径。
4. 测试与基准：新增单元测试验证 bitwise 精确匹配、拒绝不受支持输入、torch.compile(fullgraph=True) 兼容性；基准测试覆盖 14 个生产形状与 2 个 CI 轻量形状，展示 4.22x~7.34x kernel 加速。

关键文件：

- python/sglang/jit_kernel/diffusion/ltx2_qknorm_split_rope.py（模块 自定义算子；类别 source；类型 core-logic；符号 _jit_ltx2_qknorm_split_rope_module, _fake_impl, _ltx2_qknorm_split_rope_custom_op, _supported_side）：核心 Python 绑定，包含自定义 op 注册、TF 假实现、前置检查逻辑，是 CUDA 内核的入口。
- test/registered/jit/diffusion/test_ltx2_qknorm_split_rope.py（模块 测试覆盖；类别 test；类型 test-coverage；符号 _require_cuda_b200, cuda_setup, _make_cos_sin,

_apply_split_rotary_ref）：包含 bitwise 精确测试、不支持输入拒绝测试、torch.compilefullgraph 测试。

- test/registered/jit/benchmark/diffusion/bench_ltx2_qknorm_split_rope.py（模块 基准测试；类别 test；类型 benchmark；符号 Workload, _make_cos_sin, _apply_split_rotary_ref, _reference_pair）：提供生产形状与 CI 小形状基准，量化 kernel 加速效果。
- python/sglang/multimodal_gen/runtime/models/dits/ltx_2.py（模块 扩散模型；类别 source；类型 core-logic；符号 _ltx2_try_fused_qknorm_split_rope）：将 CUDA fast-path 接入 LTX2 注意力前向，包含降级逻辑。
- python/sglang/jit_kernel/csrc/diffusion/ltx2_qknorm_split_rope.cuh（模块 CUDA 内核；类别 source；类型 kernel-implementation；符号 compute_rstd, ltx2_qknorm_split_rope_kernel, LTX2QKNormSplitRopeKernel::run）：CUDA kernel 实现，包含 RMSNorm + split-RoPE 融合计算。

关键符号：ltx2_qknorm_split_rope_cuda, can_use_ltx2_qknorm_split_rope_cuda, _ltx2_try_fused_qknorm_split_rope, _supported_side, _is_sm100_or_newer, compute_rstd, ltx2_qknorm_split_rope_kernel

关键源码片段

python/sglang/jit_kernel/diffusion/ltx2_qknorm_split_rope.py

核心 Python 绑定，包含自定义 op 注册、TF 假实现、前置检查逻辑，是 CUDA 内核的入口。

```
# 从 ltx2_qknorm_split_rope.py 摘录：核心自定义 op 与前置检查

from __future__ import annotations
from typing import TYPE_CHECKING
import torch
from sglang.jit_kernel.utils import cache_once, load_jit
from sglang.srt.utils.custom_op import register_custom_op

if TYPE_CHECKING:
    from tvm_ffi.module import Module

@cache_once
def _jit_ltx2_qknorm_split_rope_module() -> Module:
    """返回 JIT 编译的 CUDA 模块（缓存）"""
    return load_jit(
        "diffusion_ltx2_qknorm_split_rope",
        cuda_files=["diffusion/ltx2_qknorm_split_rope.cuh"],
        cuda_wrappers=[
            (
                "ltx2_qknorm_split_rope_pair",
                "sglang_ltx2_qknorm_split_rope::LTX2QKNormSplitRopeKernel::run",
            )
        ],
    )
```

```
def _fake_impl(..., eps: float, num_heads: int, head_dim: int) -> tuple[torch.Tensor, torch.
Tensor]:
    """假实现，为 torch.compile fullgraph 提供形状推导"""
    return torch.empty_like(q, dtype=torch.bfloat16), torch.empty_like(k, dtype=torch.bfloat16)

@register_custom_op(
    op_name="diffusion_ltx2_qknorm_split_rope",
    mutates_args=[],
    fake_impl=_fake_impl,
)
def _ltx2_qknorm_split_rope_custom_op(..., eps: float, num_heads: int, head_dim: int) -> ...:
    """自定义 CUDA 算子入口：分配输出，调用 JIT 模块"""
    q_out = torch.empty_like(q, dtype=torch.bfloat16)
    k_out = torch.empty_like(k, dtype=torch.bfloat16)
    module = _jit_ltx2_qknorm_split_rope_module()
    module.ltx2_qknorm_split_rope_pair(q_out, k_out, q, q_cos, q_sin, q_weight, k, k_cos, k_sin,
    k_weight, float(eps), int(num_heads), int(head_dim))
    return q_out, k_out

def _supported_side(x, cos, sin, weight, *, num_heads, head_dim) -> bool:
    """检查单边（Q 或 K）的形状/布局是否满足 kernel 要求"""
    return (
        x.is_cuda and cos.is_cuda and sin.is_cuda and weight.is_cuda
        and x.device == cos.device == sin.device == weight.device
        and x.dtype == torch.bfloat16 and cos.dtype == torch.bfloat16
        and sin.dtype == torch.bfloat16 and weight.dtype == torch.bfloat16
        and x.ndim == 3 # [batch, seq, hidden]
        and cos.ndim == 4 # [batch, num_heads, seq, half_dim]
        and sin.ndim == 4
        and x.is_contiguous()
        and cos.shape == sin.shape
        and cos.shape[0] == x.shape[0] # batch 一致
        and cos.shape[1] == num_heads
        and cos.shape[2] == x.shape[1] # seq 一致
        and cos.shape[3] * 2 == head_dim # half_dim * 2 == head_dim
        and x.shape[2] == num_heads * head_dim # hidden == num_heads * head_dim
        and x.shape[2] == weight.shape[0] # weight 长度等于 hidden
        and weight.ndim == 1
        and head_dim % 2 == 0 and x.shape[2] % 4 == 0
        and cos.stride(-1) == 1 and sin.stride(-1) == 1 # 最后一维连续
    )

def _is_sm100_or_newer(x: torch.Tensor) -> bool:
    if not x.is_cuda: return False
    try:
        return torch.cuda.get_device_capability(x.device)[0] >= 10
    except RuntimeError:
        return False
```

```
def can_use_ltx2_qknorm_split_rope_cuda(q, q_cos, q_sin, q_weight, k, k_cos, k_sin, k_weight, *,
num_heads, head_dim) -> bool:
    """综合检查：SM >= 100 且 Q/K 两边形状均受支持"""
    return (
        _is_sm100_or_newer(q)
        and _supported_side(q, q_cos, q_sin, q_weight, num_heads=num_heads, head_dim=head_
        dim)
        and _supported_side(k, k_cos, k_sin, k_weight, num_heads=num_heads, head_dim=head_
        dim)
    )
```

评论区精华

gemini-code-assist[bot] 提出了若干改进建议：

- 整数类型安全：将 CUDA kernel 参数和循环变量从 int 改为 int64_t，防止大张量溢出。
- 移除冗余类型转换：对应 Python 绑定的 int(num_heads) 等转型无需，因为 TVM FFI 会处理。
- 简化 RMSNorm 检查：_ltx2_try_fused_qknorm_split_rope 中对 weight 和 eps 的 None 检查在 RMSNorm 保证下冗余。
- 代码对齐：测试文件中 _apply_split_rotary_ref 应简化以保持与基准文件一致。 所有建议均在后续 commits (sha 06a3d4cb, e9394b8) 中被采纳并关闭。
- 使用 int64_t 防止整数溢出 (correctness): 已采纳，提交 06a3d4c 中修复。
- 移除冗余类型转换与参数检查 (style): 已采纳，后续 commits 移除。

风险与影响

- 风险：
  1. 硬件限制：该路径仅在 SM≥100 (B200) 上激活，其他设备静默降级，无功能风险。
  2. JIT 编译稳定性：依赖 TVM JIT 加载 CUDA 模块；首次加载可能因环境问题失败（如缺少 CUDA 运行时），但有一次性全局禁用回退机制。
  3. 精度：单元测试覆盖了生产形状的 bitwise 相等性，精度无退化。
  4. 形状覆盖：仅验证了 head_dim=64/128、BF16 连续输入等条件，不支持的输入会回退；极端非连续 cos/sin 步长可能绕过 _supported_side 检查但未充分测试。
- 影响：
  1. 用户影响：B200 上 LTX-2.3 模型推理端到端加速 9-10%，且输出 bitwise 不变；其他用户无影响。
  2. 系统影响：新增 ~277 行 CUDA + ~205 行 Python 绑定，JIT 编译增加首次推理延迟 1-2 秒（缓存后消失）。
  3. 团队影响：引入了 KDA-Pilot 设计工具产的融合 kernel 集成模式（自定义 op + fake_impl + 降级），可被后续优化 PR 复现。 - 风险标记：B200 限定，JIT 编译依赖，自动降级

关联脉络

- 暂无明显关联 PR