PR #26318 完整报告

sgl-project/sglang

[diffusion][jit_kernel] perf: varlen FA fast path for USPAttention masked branch

合并时间：2026-05-28 21:26

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26318

执行摘要

- 一句话：Varlen FA 加速 USPAttention masked 路径，Qwen-Image 推理提速 15%+
- 推荐动作：该 PR 值得精读，尤其是对从事 Transformer inference 性能优化的工程师。核心设计模式（Triton 融合减少 launch、metadata 预计算复用、显式契约确保兼容性）具有很高的参考价值。新增的测试用例可作为 Triton 内核测试的范例。建议关注后续是否将该模式推广到其他 attention 变体（如 cross-attention、DPO 等）。

功能与动机

PyTorch SDPA 在非 None mask 下无法使用 FlashAttention，回退到 SM80 的 cutlassF 内核，在 Blackwell B200 上每个 attention 调用比原生 FA4（flash_fwd_sm100）慢约 7 倍，成为去噪循环的主要瓶颈。上游 Flash Attention 长期不支持密集 mask（Dao-AI Lab/flash-attention#409, #1990），因此需要一种 workaround 来提升 diffusion 模型（如 Qwen-Image）的推理性能。

实现拆解

1. 新增 Triton 融合 pack/scatter 内核：在 python/sglang/jit_kernel/diffusion/triton/varlen_pack_pad.py 中实现 fused_pack_qkv 和 fused_scatter_to_padded 两个融合操作。fused_pack_qkv 通过一个 Triton kernel 将 Q/K/V 按 indices gather 到连续内存（原来需要 3 次 index_select），fused_scatter_to_padded 通过另一个 Triton kernel 将 packed 输出写回 padded 布局，无效位置填 0（原来需要 zeros + index_copy_）。同时提供 build_inv_indices 辅助函数用于从 pack indices 生成反向查找表。
2. USPAttention 集成 varlen FA 快速路径：在 python/sglang/multimodal_gen/runtime/layers/attention/layer.py 中新增 build_varlen_mask_meta 函数，从 [B, S] 的 bool/int 掩码计算出 cu_seqlens、indices、inv_indices、max_seqlen 等元数据。USPAttention.forward 的 _prepare_sdpa_mask 方法中，当提供了 attn_mask_meta 且满足形状条件时，走 varlen FA 路径：先 fused_pack_qkv，再 flash_attn_varlen_func，最后 fused_scatter_to_padded。若 indices 为空则回退到全零输出。环境变量 SGLANG_VARLEN_FA 可全局禁用。
3. Qwen-Image 模型适配：在 python/sglang/multimodal_gen/runtime/models/dits/qwen_image.py 中，于 QwenImageTransformer2DModel.forward 中预计算 attn_mask_meta（调用 build_varlen_mask_meta(joint_mask)）并通过 cross_attention_kwargs 传递到每个 block，实现每请求一次 meta 计算，各 denoise block 复用。同时更新

QwenImageJointBlock.forward 接收 attn_mask_meta 参数并传给 USPAttention。

4. 导出新接口：在 python/sglang/multimodal_gen/runtime/layers/attention/__init__.py 中导出 build_varlen_mask_meta，方便外部调用。
5. 测试配套：
   - test_varlen_pack_pad.py：25 个用例覆盖 bf16/fp16、7 种生产相似形状（含 zero-text 边界）、非连续输入、空 mask，通过 torch.equal 验证融合内核与 PyTorch 参考（index_select / zeros+index_copy_）的比特一致。
   - test_varlen_uspattn_equivalence.py：端到端验证 varlen 路径与 SDPA 在有效行上对齐（FA 容差内），无效行输出精确为零。
   - 测试用例注册到 CI suite base-b-kernel-unit-1-gpu-large 和 nightly-kernel-1-gpu。

关键文件：

- python/sglang/jit_kernel/diffusion/triton/varlen_pack_pad.py（模块 JIT 内核；类别 source；类型 core-logic；符号 _fused_pack_qkv_kernel, fused_pack_qkv, _fused_scatter_to_padded_kernel, fused_scatter_to_padded）：核心变更文件，新增两个融合 Triton 内核（pack 和 scatter），替代 5 次 PyTorch launch，是性能优化的核心。
- python/sglang/multimodal_gen/runtime/layers/attention/layer.py（模块 注意力层；类别 source；类型 core-logic；符号 build_varlen_mask_meta）：修改文件：新增 build_varlen_mask_meta 函数和 USPAttention 的 varlen FA 快速路径，是整个优化在模型层面的集成点。
- python/sglang/multimodal_gen/runtime/models/dits/qwen_image.py（模块 模型逻辑；类别 source；类型 data-contract）：修改文件：在 QwenImageTransformer2DModel 中预计算 attn_mask_meta 并传递给每个 block，是实际触发优化的调用方。
- python/sglang/jit_kernel/tests/diffusion/test_varlen_pack_pad.py（模块 测试；类别 test；类型 test-coverage；符号 _build_mask, _ref_pack, _ref_scatter, _build_meta）：新增测试文件：25 个用例验证融合内核与 PyTorch 参考的比特一致性，覆盖多种形状和边界条件。
- python/sglang/jit_kernel/tests/diffusion/test_varlen_uspattn_equivalence.py（模块 测试；类别 test；类型 test-coverage；符号 _build_mask, _sdpa_with_key_mask, _varlen_path, test_varlen_path_matches_sdpa_on_valid_rows）：新增测试文件：端到端验证 varlen 路径在有效行上匹配 SDPA（容差内），无效行精确为零。
- python/sglang/multimodal_gen/runtime/layers/attention/__init__.py（模块 注意力层；类别 source；类型 core-logic）：修改文件：导出 build_varlen_mask_meta，供外部模块使用。

关键符号：build_varlen_mask_meta, fused_pack_qkv, fused_scatter_to_padded, build_inv_indices, _fused_pack_qkv_kernel, _fused_scatter_to_padded_kernel, USPAttention.forward, _prepare_sdpa_mask, QwenImageJointBlock.forward, QwenImageTransformer2DModel.forward

关键源码片段

python/sglang/jit_kernel/diffusion/triton/varlen_pack_pad.py

核心变更文件，新增两个融合 Triton 内核（pack 和 scatter），替代 5 次 PyTorch launch，是性能优化的核心。

```
"""Fused Triton pack/scatter kernels for the varlen mask path.

Used by ``USPAttention.forward`` masked branch to gather Q/K/V at valid
positions and scatter the FA output back to the dense ``[B, S, H, D]`` layout.
"""

from __future__ import annotations

import torch
import triton
import triton.language as tl


@triton.jit
def _fused_pack_qkv_kernel(
    Q_ptr, K_ptr, V_ptr,
    Q_unpad_ptr, K_unpad_ptr, V_unpad_ptr,
    indices_ptr,
    HD, # H * D, 展平后的特征维度
    src_row_stride, # Q/K/V 中行之间的步长（B*S 行的下一行）
    dst_row_stride, # Q_unpad/K_unpad/V_unpad 中的行步长
    BLOCK_HD: tl.constexpr,
):
    """每个 packed 行一个程序；将 Q[src]、K[src]、V[src] 复制到目标行。"""
    out_row = tl.program_id(0)
    src_row = tl.load(indices_ptr + out_row).to(tl.int64)
    cols = tl.arange(0, BLOCK_HD)
    col_mask = cols < HD
    src_offset = src_row * src_row_stride + cols
    dst_offset = out_row * dst_row_stride + cols
    # 一次性加载 Q、K、V
    q_val = tl.load(Q_ptr + src_offset, mask=col_mask)
    k_val = tl.load(K_ptr + src_offset, mask=col_mask)
    v_val = tl.load(V_ptr + src_offset, mask=col_mask)
    tl.store(Q_unpad_ptr + dst_offset, q_val, mask=col_mask)
    tl.store(K_unpad_ptr + dst_offset, k_val, mask=col_mask)
    tl.store(V_unpad_ptr + dst_offset, v_val, mask=col_mask)


def fused_pack_qkv(
    q: torch.Tensor, k: torch.Tensor, v: torch.Tensor, indices: torch.Tensor,
) -> tuple[torch.Tensor, torch.Tensor, torch.Tensor]:
    """将 ``[B, S, H, D]`` 的 Q/K/V 按 ``indices`` 打包成 ``[total_valid, H, D]``。

    ``indices`` 是 int64 类型的扁平 ``B*S`` 位置，指示每个保留 token 的位置。
    非连续输入会在内部转换为连续。
```

```
    """
    assert q.shape == k.shape == v.shape, "Q/K/V 形状必须一致"
    assert q.dtype == k.dtype == v.dtype, "Q/K/V 数据类型必须一致"
    assert q.dim() == 4, "Q/K/V 必须是 [B, S, H, D]"
    assert indices.dtype in (torch.int32, torch.int64)
    q = q.contiguous()
    k = k.contiguous()
    v = v.contiguous()
    bs, seq, num_heads, head_dim = q.shape
    hd = num_heads * head_dim
    n_valid = indices.shape[0]
    # 空 mask 保护：FA varlen 不接受零长度输入
    if n_valid == 0:
        return (q.new_empty(0, num_heads, head_dim),
                k.new_empty(0, num_heads, head_dim),
                v.new_empty(0, num_heads, head_dim))
    block_hd = triton.next_power_of_2(hd)
    q_flat = q.view(bs * seq, hd)
    k_flat = k.view(bs * seq, hd)
    v_flat = v.view(bs * seq, hd)
    q_unpad = torch.empty(n_valid, hd, dtype=q.dtype, device=q.device)
    k_unpad = torch.empty(n_valid, hd, dtype=k.dtype, device=k.device)
    v_unpad = torch.empty(n_valid, hd, dtype=v.dtype, device=v.device)
    with torch.get_device_module().device(q.device):
        _fused_pack_qkv_kernel[(n_valid,)](
            q_flat, k_flat, v_flat,
            q_unpad, k_unpad, v_unpad,
            indices,
            hd, q_flat.stride(0), q_unpad.stride(0),
            BLOCK_HD=block_hd,
        )
    return (q_unpad.view(n_valid, num_heads, head_dim),
            k_unpad.view(n_valid, num_heads, head_dim),
            v_unpad.view(n_valid, num_heads, head_dim))
```

python/sglang/multimodal_gen/runtime/layers/attention/layer.py

修改文件：新增 build_varlen_mask_meta 函数和 USPAttention 的 varlen FA 快速路径，是整个优化在模型层面的集成点。

```
import os
from sglang.jit_kernel.diffusion.triton.varlen_pack_pad import (
    build_inv_indices, fused_pack_qkv, fused_scatter_to_padded,
)
from sglang.jit_kernel.flash_attention import flash_attn_varlen_func
from sglang.multimodal_gen.runtime.layers.attention.backends import (
    flash_attn as _fa_backend,
)

# 环境变量开关：设置为 "0" 禁用 varlen FA 快速路径
```

```
_VARLEN_FA_ENABLED = os.environ.get("SGLANG_VARLEN_FA", "1") != "0"


def build_varlen_mask_meta(key_mask: torch.Tensor) -> dict:
    """从 ``[B, S]`` 的 key mask 构建 varlen FA 元数据。

    返回 ``cu_seqlens``、``indices``、``inv_indices``、``max_seqlen``。
    将结果通过 ``joint_attention_kwargs`` 传入可启用 USPAttention 的 varlen
    FA 快速路径，该路径会将 masked query 行的输出置零——仅当下游丢弃或忽略
    这些行时才能使用。
    """
    assert key_mask.dim() == 2, "key_mask 必须为 [B, S]"
    bs, seq = key_mask.shape
    bool_mask = key_mask.to(dtype=torch.bool)
    valid_lens = bool_mask.sum(dim=1, dtype=torch.int32)
    # 展平后的有效位置索引
    indices = bool_mask.reshape(-1).nonzero(as_tuple=False).flatten()
    cu_seqlens = torch.zeros(bs + 1, dtype=torch.int32, device=key_mask.device)
    cu_seqlens[1:] = torch.cumsum(valid_lens, dim=0)
    inv_indices = build_inv_indices(indices, bs * seq)
    return {
        "cu_seqlens": cu_seqlens,
        "indices": indices,
        "inv_indices": inv_indices,
        "max_seqlen": seq, # 上界；FA varlen 实际使用 cu_seqlens 确定范围
    }


# 在 USPAttention.forward 中（_prepare_sdpa_mask 方法）：
if attn_mask_meta is not None:
    # 快速路径：使用 varlen FA
    indices = attn_mask_meta["indices"]
    if indices.shape[0] == 0:
        # 全 False mask，直接返回全零
        return torch.zeros_like(q)
    q_unp, k_unp, v_unp = fused_pack_qkv(q, k, v, indices)
    out_unp = flash_attn_varlen_func(
        q=q_unp, k=k_unp, v=v_unp,
        cu_seqlens_q=attn_mask_meta["cu_seqlens"],
        cu_seqlens_k=attn_mask_meta["cu_seqlens"],
        max_seqlen_q=attn_mask_meta["max_seqlen"],
        max_seqlen_k=attn_mask_meta["max_seqlen"],
        softmax_scale=softmax_scale,
        causal=False,
        ver=_fa_backend.fa_ver,
    )
    return fused_scatter_to_padded(out_unp, attn_mask_meta["inv_indices"], bs, seq)
```

python/sglang/multimodal_gen/runtime/models/dits/qwen_image.py

修改文件：在 QwenImageTransformer2DModel 中预计算 attn_mask_meta 并传递给每个 block，是实际触发优化的调用方。

```
from sglang.multimodal_gen.runtime.layers.attention import (
    USPAttention,
    build_varlen_mask_meta,
)

# 在 QwenImageTransformer2DModel.forward 中，构建 joint mask 后：
joint_mask = torch.cat([encoder_hidden_states_mask, image_mask], dim=1)
block_attention_kwargs["attn_mask"] = joint_mask
# 预计算 varlen 元数据，每请求只计算一次，所有 block 复用
block_attention_kwargs["attn_mask_meta"] = build_varlen_mask_meta(joint_mask)

# 在 QwenImageJointBlock.forward 中接收并传递：
attn_mask_meta = cross_attention_kwargs.get("attn_mask_meta")
out = usp_attn(
    query, key, value,
    attn_mask=attn_mask,
    attn_mask_meta=attn_mask_meta, # 传入 varlen 元数据
    num_replicated_prefix=seq_len_txt,
)
```

评论区精华

- 语义变更担忧：BBuf 指出 varlen 路径将 masked query rows 的输出置零，与 SDPA 的 key mask 语义（mask 仅作用于 key，query 保持）不同，对于其他使用 USPAttention 的调用者可能存在风险。讨论结果是：不再从 SDPA 内部推断，而是要求调用者显式提供 attn_mask_meta（通过 build_varlen_mask_meta 构建），明确接受“masked rows 将被 drop 并置零”的契约。若不提供，则回退到 SDPA，确保默认行为不变。
- 端到端测试要求：BBuf 要求增加对比 varlen 路径与 SDPA 的等价性测试。作者补充了 test_varlen_uspattn_equivalence.py，验证有效行 match SDPA（FA 容差内），无效行精确为零。
- 设备一致性检查：BBuf 建议添加 attn_mask.device == q.device 断言。作者在对应的 commit 中已添加此检查。
- 环境变量开关：提供 SGLANG_VARLEN_FA=0 可完全回退到 SDPA，确保安全降级。
- 语义变更：masked query rows 被置零 vs key mask 语义 (design): 采用显式契约：调用者必须通过 build_varlen_mask_meta 构建 attn_mask_meta 并传入，明确接受“masked rows 被 drop 并置零”的语义。非 mask 路径回退到 SDPA，不影响已有行为。
- 建议添加 varlen 路径与 SDPA 的端到端等价测试 (testing): 作者新增 test_varlen_uspattn_equivalence.py，验证有效行在 FA 容差范围内匹配 SDPA，无效行精确为零。
- 设备一致性检查建议 (correctness): 作者在后续提交中添加了此项检查（PR #26318 的 commit 中体现）。

风险与影响

- 风险：
  - 语义改变风险：masked query rows 输出被置零而非 SDPA 的正常注意力输出。如果下游需要这些行的注意力输出，将导致错误。已通过显式 attn_mask_meta 契约和文档说明来缓解，且仅在传入 meta 时生效，不影响已有调用者。
  - 兼容性风险：varlen 路径严格限制在 bool/int 2D [B, S] mask、Q/K/V shape 一致、cuda 设备、bf16/fp16。float 相加 mask 和 cross-attention 形状保持原有 SDPA 路径，不会错误应用优化。
  - 数值差异风险：FlashAttention 与 SDPA 存在数值容差（bf16 下 rtol=1e-2, atol=5e-2），测试已验证有效行在容差内。但某些极端 case 可能超出容差，需关注。
  - 回归风险：非 mask 模型（FLUX.2-klein-4B）经测试零回归，且优化路径本就不会触发。环境变量可全局关闭。
- 影响：
  - 用户影响：使用 Qwen-Image 等 diffusion 模型的用户将获得 14-21% 的推理加速（饱和批），显著降低去噪循环延迟。其他模型用户不受影响。
  - 系统影响：新增 Triton 内核编译开销（首次运行时），但运行后无额外开销。减少 GPU launch 次数，对整体系统负载有正面影响。
  - 团队影响：提供了一种可重复的模式（build_varlen_mask_meta + 融合 pack/scatter），未来可推广到其他需要 mask 的 attention 场景。
  - 风险标记：语义变更（masked rows zero-filled），仅适用于 bool/int 2D mask, FA vs SDPA 数值容差，依赖环境变量控制

关联脉络

- PR #24737 Support Flashinfer Cute-DSL MLA attention: 同样属于 attention 性能优化，利用硬件特性（Blackwell）加速 decoding，与当前 PR 共享 attention 性能优化的目标。
- PR #26513 Fix FlashInfer SWA EXTEND-with-prefix correctness in merge_state path: 均为 attention 后端的 bugfix/ 优化，反映团队在 attention 性能与正确性方面的系统性工作。