PR #20479 完整报告

sgl-project/sglang

Support Triton MLA FP8 KV cache

合并时间：2026-05-07 09:32

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/20479

执行摘要

- 一句话：支持 Triton MLA FP8 KV 缓存，长序列性能提升 91%
- 推荐动作：值得所有关注 MLA 和 Triton 内核优化的工程师精读。特别是 v = tl.trans(k) 技巧、KV Splits 的动态计算、以及 PDL 的使用都是可以直接复用到其他模型的优化模式。建议在后续 PR 中补充单元测试和 k_scale==v_scale 的检查。

功能与动机

在 SM120 设备上，Flashinfer 不支持 FP8 Attention，而 XQA 仅支持有限的 head dim（如 DeepSeek-R1 的 head dim，但不支持 Kimi 等模型）。此前 Triton 后端在 MLA 和长序列场景下性能极差。因此需要扩展 Triton MLA 后端以支持 FP8 KV 缓存，使其成为 SM120 上 MLA 模型的可行方案。

实现拆解

1. 新增 KV Splits 上限计算函数：在 triton_backend.py 中定义 _mla_decode_kv_splits_cap(base_max_kv_splits, sm_count, max_context_len)，该函数根据 SM 数量和最大上下文长度计算一个更合理的 KV 拆分上限，使长上下文请求能充分利用所有 SM，同时避免短上下文时静态缓冲区过大。
2. 重构解码内核以支持 MLA 和 PDL：在 decode_attention.py 的 _fwd_grouped_kernel_stage1 中添加 HAS_MLA 和 USE_PDL 编译时常量。当 HAS_MLA 时，V 通过 tl.trans(k) 从 K 导出（因为 MLA 中 K 和 V 共享存储），从而省去独立的 V 加载和后续点积。同时将循环不变计算（如 base_offs_k）提到循环外，并将 q 的类型转换提前到循环外。USE_PDL 启用时，在 Stage1 结尾调用 tl.extra.cuda.gdc_launch_dependents() 以实现异步启动。
3. 在 forward 方法中处理 FP8 缩放：在 forward_decode 和 forward_extend 中，当使用 MLA 且 layer.k_scale 不为 None 时，手动将 k 除以 k_scale 后存入 KV 缓存池。在 forward_extend 中需要 clone 以避免影响后续计算，在 forward_decode 中 k 不再使用因此原位缩放。
4. 清理扩展内核中的冗余转换：在 extend_attention.py 中移除 q.to(k.dtype) 的中间变量，因为 tl.dot 会自动处理类型提升，且已在循环外正确转换。
5. 启用 PDL 检测：在 TritonAttnBackend.__init__ 中通过 sgl_kernel.utils.is_arch_support_pdl() 检测当前架构是否支持 PDL，并在调用解码内核时

传递 use_pdl 参数。

关键文件：

- python/sglang/srt/layers/attention/triton_backend.py（模块 注意力后端；类别 source；类型 core-logic；符号 _mla_decode_kv_splits_cap）：核心调度文件：新增 MLA FP8 KV 缓存支持，包含 KV splits 上限计算和 PDL 启用逻辑，以及 forward_decode/extend 中缩放处理。
- python/sglang/srt/layers/attention/triton_ops/decode_attention.py（模块 解码内核；类别 source；类型 core-logic）：解码内核实现：引入 HAS_MLA 和 USE_PDL 编译时常量，优化循环结构，是性能提升的核心。
- python/sglang/srt/layers/attention/triton_ops/extend_attention.py（模块 扩展内核；类别 source；类型 refactor）：辅助调整：移除冗余类型转换，保持代码整洁。

关键符号：_mla_decode_kv_splits_cap, TritonAttnBackend.forward_decode, TritonAttnBackend.forward_extend, _fwd_grouped_kernel_stage1, _decode_grouped_att_m_fwd

关键源码片段

python/sglang/srt/layers/attention/triton_backend.py

核心调度文件：新增 MLA FP8 KV 缓存支持，包含 KV splits 上限计算和 PDL 启用逻辑，以及 forward_decode/extend 中缩放处理。

```
# 将 KV splits 上限提升到至少覆盖 min(sm_count, max_context_len / 32) 的 2 的幂
# 这样长上下文可以充分利用所有 SM，短上下文则不会分配过多静态缓冲区
_MLA_DECODE_MIN_BLOCK_KV = 32

def _mla_decode_kv_splits_cap(
    base_max_kv_splits: int, sm_count: int, max_context_len: int
) -> int:
    if sm_count <= 0:
        return base_max_kv_splits
    sm_cap = next_power_of_2(sm_count)
    ctx_cap = next_power_of_2(triton.cdiv(max_context_len, _MLA_DECODE_MIN_BLOCK_KV))
    return max(base_max_kv_splits, min(sm_cap, ctx_cap))

# 在 __init__ 中调用
if self.use_mla:
    self.max_kv_splits = _mla_decode_kv_splits_cap(
        self.max_kv_splits,
        self.device_core_count,
        self.max_context_len,
    )
self.use_pdl = is_arch_support_pdl()
```

python/sglang/srt/layers/attention/triton_ops/decode_attention.py

解码内核实现：引入 HAS_MLA 和 USE_PDL 编译时常量，优化循环结构，是性能提升的核心。

```
# 循环前：提升循环不变计算
base_offs_k = cur_kv_head * stride_buf_kh + offs_d[:, None]
if BLOCK_DPE > 0:
    base_offs_kpe = cur_kv_head * stride_buf_kh + offs_dpe[:, None]
if not HAS_MLA:
    base_offs_v = cur_kv_head * stride_buf_vh + offs_dv[None, :]

q = tl.load(Q + offs_q, mask=..., other=0.0)
q_k = q.to(K_Buffer.dtype.element_ty) # 类型转换移出循环

for start_n in tl.range(split_kv_start, split_kv_end, BLOCK_N):
    offs_n = start_n + tl.arange(0, BLOCK_N)
    kv_loc = ...
    offs_buf_k = kv_loc[None, :] * stride_buf_kbs + base_offs_k
    k = tl.load(K_Buffer + offs_buf_k, mask=..., other=0.0)
    qk = tl.dot(q_k, k) # 不再重复转换

    if BLOCK_DPE > 0:
        offs_buf_kpe = kv_loc[None, :] * stride_buf_kbs + base_offs_kpe
        kpe = tl.load(...)
        qk += tl.dot(qpe.to(kpe.dtype), kpe)

    if HAS_MLA:
        # MLA 中 K 和 V 是同一个张量，直接转置 K 得到 V
        v = tl.trans(k)
    else:
        offs_buf_v = kv_loc[:, None] * stride_buf_vbs + base_offs_v
        v = tl.load(V_Buffer + offs_buf_v, mask=..., other=0.0)

    # ... 后续 online softmax 和累加 ...

if USE_PDL:
    tl.extra.cuda.gdc_launch_dependents() # 程序化依赖启动
```

评论区精华

重点关注以下讨论：

- K_scale 安全性（mmangkad）：指出只有当 k_scale == v_scale 时预缩放 K 才正确，建议添加断言或统一反量化。作者回应认为可后续跟进。
- 复用 is_arch_support_pdl（Qiaolin-Yu）：建议使用已有的 is_arch_support_pdl 而非新增 supports_pdl。作者采纳。
- KV splits cap 边界（alexnails）：建议加入 max_context_len 上限以防止短上下文支付过高静态成本。作者采纳并修改实现。
- 循环优化（alexnails）：建议将 q.to(k.dtype) 移出循环、使用 tl.range 并指定 num_stages 以提升性能。作者均已采纳。
- K scaling safety (correctness): 作者认可但认为可后续跟进，未添加断言。

- Reuse existing PDL check function (design): 作者改用 is_arch_support_pdl。
- KV splits cap boundary (design): 作者采纳并添加 max_context_len 参数。
- Move type cast out of loop (performance): 作者将 q_k 类型转换后复用。
- Use tl.range with num_stages (performance): 作者使用 tl.range(..., num_stages=2)。

风险与影响

- 风险：主要风险：
  1. K 缩放假设：若模型 k_scale != v_scale，手动缩放 K 后再存入（未缩放 V）可能导致后续注意力计算错误。当前代码假设二者相等，但未显式断言。
  2. 无对应测试：本 PR 未新增测试用例，FP8 MLA 的精度和正确性缺乏自动化验证，尤其长序列场景。
  3. 架构依赖：PDL 支持基于 SM 版本，但 is_arch_support_pdl 对 SM9.0+ 返回 True，对于未测试的架构可能存在稳定性风险。
  4. 短上下文影响：_mla_decode_kv_splits_cap 提升了 KV 拆分下限，对于极短上下文可能增加少量内核启动开销，但已通过 max_context_len 边界缓解。 - 影响：对用户而言，使用 --kv-cache-dtype fp8_e4m3 --attention-backend triton 且模型使用 MLA 时（如 Kimi K2.5、DeepSeek-R1），在 SM120 设备上将获得显著的解码速度提升（如 196K 上下文 TPOT 从 213ms 降至 19ms）。对其他架构（如 SM90）无直接影响，但若使用 FP8+MLA 会启用此优化。对系统无依赖变更。对团队而言，此 PR 展示了在 Triton 内核中利用 MLA 数据结构特点（KV 共享）和现代 CUDA 特性（PDL）进行深度优化的模式。 - 风险标记：k_scale 一致性假设，缺少测试覆盖，仅限 SM120 验证

关联脉络

- 暂无明显关联 PR