

PR #38687 完整报告

sgl-project/sglang

[Kernel] Add OOT dispatch for clamp position

合并时间: 2026-09-15 10:22

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/38687>

执行摘要

- 一句话: clamp_position 接入 fused-op, 新增 OOT 分发入口
- 推荐动作: 值得快速精读 python/sglang/kernels/ops/attention/clamp_position.py, 学习 SGLang 的 BaseFusedOp/register_fused_op OOT 扩展模式, 重点看 priority、capabilities、FormatSignature 如何决定后端分发; 外部硬件接入时可仿照本 PR 使用 register_oop_forward("attention.clamp_position", ...)。同时注意公共 wrapper 的 int64 输出契约是刻意设计, 接入 int32 buffer 的调用方需要自行适配。

功能与动机

PR body 明确指出问题: clamp_position 目前通过 import-time 平台检查选择 CUDA/HIP 实现, OOT 硬件插件无法通过 SGLang 的 typed fused-op dispatch 注册优化实现, 只能替换模块级函数或挂通用 plugin hook。本次改造把实现选择下沉到 BaseFusedOp 的 capabilities/priority 机制, 让外部平台可以正规注册自己的 fused 实现, 而不是替换核心 SGLang 函数。

实现拆解

1. 新增 fused-op 封装: 在 python/sglang/kernels/ops/attention/clamp_position.py 引入 BaseFusedOp、register_fused_op、KernelBackend、CapabilityRequirement、FormatSignature。新增 ClampPositionOp 类: op = "attention.clamp_position", priority 为 JIT 优先、TORCH 兜底, capabilities 声明 JIT 仅支持 CUDA/HIP; forward_native 实现原生 torch.clamp(seq_lens - 1, min=0).to(torch.int64), forward_jit 复用原 clamp_position_cuda 并补 .to(torch.int64)。随后用 register_fused_op 注册单例, 并暴露公共 clamp_position() wrapper。
2. ForwardBatch 调用点改造: 在 python/sglang/srt/model_executor/forward_batch_info.py 删除本地 _clamp_position_native 与 if is_cuda() or is_hip() 分支, 移除 is_cuda 导入, 改为直接从 kernels 公共入口导入 clamp_position。平台选择完全下沉到 fused-op dispatch, forward-batch 路径继续计算 clamp(seq_lens - 1, min=0) 并输出 int64。
3. 测试配套: test/registered/kernels/ops/attention/test_clamp_position.py 保留原 clamp_position_cuda 的 dtype-preserving 测试, 新增 _reference_public_clamp_position (公共契约固定 int64 输出) 以及 test_native (CPU 上直接调 forward_native) 和 test_public_dispatch (CUDA 上调公共 wrapper), 分别用 int32/int64 输入验证。

4. CI 与验证：无配置、schema 或部署改动；作者说明 PyTorch 单测未在本地执行，由 run-ci、run-ci-extra 以及 AMD ROCm runner 的 CI 完成验证。

关键文件：

- python/sglang/kernels/ops/attention/clamp_position.py（模块 内核算子；类别 infra；类型 infrastructure；符号 ClampPositionOp, forward_native, forward_jit, clamp_position）：核心改造文件：把原有 CUDA/HIP JIT kernel 包装为 ClampPositionOp，注册为 attention.clamp_position，并新增公共 clamp_position() wrapper，是 OOT 扩展点的载体。
- python/sglang/srt/model_executor/forward_batch_info.py（模块 批处理；类别 source；类型 data-contract；符号 clamp_position, _clamp_position_native）：核心调用路径：删除 import-time 平台分支，统一改走公共 wrapper，是本次重构在下游的唯一实际调用点。
- test/registered/kernels/ops/attention/test_clamp_position.py（模块 单元测试；类别 test；类型 test-coverage；符号 _reference_public_clamp_position, test_native, test_public_dispatch）：新增 native 与公共分发路径的 int32/int64 测试，明确公共 wrapper 固定输出 int64 的契约。

关键符号：ClampPositionOp.forward_native, ClampPositionOp.forward_jit, clamp_position, clamp_position_cuda

关键源码片段

python/sglang/kernels/ops/attention/clamp_position.py

核心改造文件：把原有 CUDA/HIP JIT kernel 包装为 ClampPositionOp，注册为 attention.clamp_position，并新增公共 clamp_position() wrapper，是 OOT 扩展点的载体。

```
import torch

from sglang.kernels.fused_op import BaseFusedOp, register_fused_op
from sglang.kernels.spec import CapabilityRequirement, FormatSignature, KernelBackend

class ClampPositionOp(BaseFusedOp):
    """Compute non-negative, zero-based decode positions.

    OOT 硬件可以通过 register_oop_forward() 以 "attention.clamp_position"
    为目标覆盖默认实现，无需替换 forward_batch_info 里的模块级函数。
    """

    op = "attention.clamp_position"
    # JIT 优先，TORCH 作为无 JIT 平台的兜底
    priority = (KernelBackend.JIT, KernelBackend.TORCH)
    capabilities = {
        # 只有 CUDA/HIP 声明支持默认 JIT kernel，CPU/NPU 等会回落到 native
        KernelBackend.JIT: frozenset(
            {CapabilityRequirement.CUDA, CapabilityRequirement.HIP}
        )
    }
```

```

format_signature = FormatSignature(
    supported_dtypes=("int32", "int64"),
    description="clamp(seq_lens - 1, min=0); returns int64 positions",
)

def forward_native(self, seq_lens: torch.Tensor) -> torch.Tensor:
    # 与旧 _clamp_position_native 语义一致，输出固定为 int64
    return torch.clamp(seq_lens - 1, min=0).to(torch.int64)

def forward_jit(self, seq_lens: torch.Tensor) -> torch.Tensor:
    # 保留原 JIT kernel，并显式把输出统一为 int64 (ForwardBatch 依赖该 dtype)
    return clamp_position_cuda(seq_lens).to(torch.int64)

# 注册为全局 fused-op 单例，之后的公共调用都经过 dispatch
_CLAMP_POSITION = register_fused_op(ClampPositionOp(), __name__, "_CLAMP_POSITION")

def clamp_position(seq_lens: torch.Tensor) -> torch.Tensor:
    """Public wrapper: CUDA/HIP 走 JIT，其余平台走 native，OOT 可整体替换。"""
    return _CLAMP_POSITION(seq_lens)

```

test/registered/kernels/ops/attention/test_clamp_position.py

新增 native 与公共分发路径的 int32/int64 测试，明确公共 wrapper 固定输出 int64 的契约。

```

# test_clamp_position.py 新增部分：验证公共 wrapper 的 int64 输出契约
def _reference_public_clamp_position(seq_lens):
    # 公共 wrapper 的契约：无论输入是 int32 还是 int64，输出都是 int64
    return torch.clamp(seq_lens - 1, min=0).to(torch.int64)

@pytest.mark.parametrize("dtype", [torch.int32, torch.int64])
def test_native(dtype):
    seq_lens = torch.tensor([0, 1, 5, 128], dtype=dtype)
    # CPU 侧直接使用 native 实现，也必须满足 int64 输出契约
    result = ClampPositionOp().forward_native(seq_lens)
    assert result.dtype == torch.int64
    assert torch.equal(result, _reference_public_clamp_position(seq_lens))

@pytest.mark.parametrize("dtype", [torch.int32, torch.int64])
def test_public_dispatch(dtype):
    seq_lens = torch.tensor([0, 1, 5, 128], dtype=dtype, device="cuda")
    # CUDA 侧走 JIT dispatch，同样验证 int64 dtype 契约
    result = clamp_position(seq_lens)
    assert result.dtype == torch.int64
    assert torch.equal(result, _reference_public_clamp_position(seq_lens))

```

评论区精华

该 PR 没有任何 review 评论，BBuf 直接 APPROVED，合入过程没有出现设计争议或未解决疑虑。唯一值得提炼的信息来自 PR body：作者明确说明 PyTorch 单测未在本地执行（本机环境无 PyTorch/pytest），行为正确性依赖 CI；body 中记录的 CI 状态并非全绿。因此合入决策主要基于“改动小、语义明确”，并以 CI 作为主要验证手段。

- 无 review 评论，仅 BBuf APPROVED (other)：无未解决讨论项，合入决策主要基于改动小、语义明确，并由 CI 承担验证。

风险与影响

- 风险：
 - 输出 dtype 契约变化：旧代码在 CUDA/HIP 下 clamp_position 直接指向 clamp_position_cuda，输出 dtype 与输入一致 (torch.empty_like)；新公共 wrapper 固定输出 int64。仓库内 ForwardBatch 路径预期保持 int64，但任何在 forward_batch_info.py 之外直接使用旧模块级 clamp_position 且输入为 int32 的调用方会观察到 dtype 变化。
 - 核心路径变更：forward_batch_info.py 是每次 forward 构造位置信息的核心路径，现在每次调用都要经过 fused-op dispatch；若 dispatch 在某个后端（如 NPU、CPU）的能力判断出现偏差，回退行为取决于 priority/capabilities 的默认语义，而旧代码的 CPU/native 分支是显式的。
 - 测试覆盖依赖 CI：作者没有本地执行 PyTorch 单测；新增 test_public_dispatch 覆盖 CUDA 路径，AMD CI 会跑同一测试，但 OOT 插件场景无法在仓库内 CI 覆盖。
 - 性能风险低：CUDA/HIP 仍走原 JIT kernel，无内核改动，dispatch 额外开销可忽略。
- 影响：
 - 对用户：无模型输出变化，forward-batch 路径行为保持一致（仍为 int64 输出）。
 - 对 OOT 硬件厂商：获得稳定扩展点 attention.clamp_position，可通过 register_oot_forward 替换 clamp-position 实现，不再依赖 import hook 或模块级函数替换。
 - 对团队 / 维护者：为后续更多简单 kernel 迁移到 typed fused-op 注册体系提供了可复制的样板，降低 OOT 适配成本。
 - 影响范围：中等，改动集中在位置准备与 kernel 注册，不涉及调度、KV cache 或模型权重。
 - 风险标记：核心路径变更，输出 dtype 契约变化，单测依赖 CI 验证

关联脉络

- PR #39446 Reland fix(qsa): clamp the compress gather to the rows (#38346): 同为 attention 路径中的 clamp 语义修正，操作对象不同但共享 clamp 语义与注意力内核测试域。
- PR #38554 [Spec] Allow speculative workers to stage prefill shared reads: 同为默认不改变上游行为的扩展机制，与 OOT 注册思路在调度 /worker 侧平行。