

PR #43994 完整报告

vllm-project/vllm

[Kernel][Helion][1/N] Add Helion kernel for silu_and_mul_per_block_quant

合并时间: 2026-07-07 00:19

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/43994>

执行摘要

- 一句话: 新增 silu_and_mul_per_block_quant 的 Helion 内核
- 推荐动作: 值得关注该 PR 中建立的 Helion 内核集成模式: 通过 register_kernel 注册、CaseKey 配置选择、生成输入与基线对比。对于需要极致性能的团队, 可参考此路径定制融合内核。建议后续集成到模型前向中 (如替代 silu_and_mul+per_block_quant), 以释放端到端收益。建议关注 issue #32962 中后续 Helion 内核的进度。

功能与动机

作为 Helion 内核集成项目 (issue #32962) 的子任务, 目标是通过自定义内核提升 silu_and_mul_per_block_quant 操作的性能, 该操作是模型量化中的常见融合算子。

实现拆解

1. 在 vllm/kernels/helion/ops/silu_and_mul_per_block_quant.py 中实现 Helion 内核, 使用 @register_kernel 装饰器注册, 使内核可作为 vLLM CustomOp 调用。内核函数 silu_and_mul_per_block_quant 使用 Helion DSL 定义张量计算和逐块量化逻辑。
2. 实现辅助函数: generate_inputs 生成多种形状 (num_tokens, intermediate_size, group_size) 的输入, 用于自动调优; pick_config 实现分层最近邻配置选择策略 (先匹配 intermediate_size, 再匹配 group_size, 最后选择最小的满足 num_tokens ≥ 输入的配置, 若全大于则回退到最大); fake_impl 和 baseline 提供参考实现用于正确性对比。
3. 在 vllm/kernels/helion/configs/silu_and_mul_per_block_quant/ 下添加 H100 和 B200 的预调优 JSON 配置。每个配置包含 key (中间尺寸、分组大小、token 数) 和 config (编译器参数如 block_sizes, loop_orders, num_warps, num_stages, indexing 策略等)。
4. 在 tests/kernels/helion/test_silu_and_mul_per_block_quant.py 中添加单元测试。配置选择器测试覆盖精确匹配、最近匹配、无配置回退和最大形状回退。正确性测试使用 pytest.mark.parametrize 遍历多种形状, 对比内核输出与 baseline 的绝对误差 (设置 random_seed 确保可复现)。
5. 根据 review 反馈完善测试: 补全 is_scale_transposed 参数测试, 改进 scale_ub 生成策略, 并移除内核中关于 is_scale_transposed 的断言, 使其成为 dummy 参数以匹配 torch.ops._C 接口, 简化未来 vLLM 集成。

关键文件:

- `vllm/kernels/helion/ops/silu_and_mul_per_block_quant.py` (模块 内核层; 类别 `source`; 类型 `core-logic`; 符号 `silu_and_mul_per_block_quant`, `baseline`, `pick_config`, `generate_inputs`) : Helion 内核的核心实现, 包含融合算子、配置选择逻辑和基准测试函数。
- `tests/kernels/helion/test_silu_and_mul_per_block_quant.py` (模块 测试; 类别 `test`; 类型 `test-coverage`; 符号 `TestSiluAndMulPerBlockQuantConfigPicker`, `test_config_picker_exact_match`, `test_config_picker_closest_match`, `test_config_picker_no_configs`) : 验证内核正确性和配置选择逻辑, 包含多种形状和精度测试。
- `vllm/kernels/helion/configs/silu_and_mul_per_block_quant/nvidia_h100.json` (模块 配置; 类别 `config`; 类型 `configuration`) : H100 预调优配置, 涵盖多种 `intermediate_size/num_tokens` 组合, 共 2200+ 行。
- `vllm/kernels/helion/configs/silu_and_mul_per_block_quant/nvidia_b200.json` (模块 配置; 类别 `config`; 类型 `configuration`) : B200 预调优配置, 类似但覆盖较少形状。

关键符号: `silu_and_mul_per_block_quant`, `baseline`, `pick_config`, `generate_inputs`, `fake_impl`

关键源码片段

`vllm/kernels/helion/ops/silu_and_mul_per_block_quant.py`

Helion 内核的核心实现, 包含融合算子、配置选择逻辑和基准测试函数。

缓存已选择的配置, 避免重复计算

`_pick_cache: dict[tuple[int, int, int], CaseKey | None] = {}`

`def pick_config(args: tuple[Any, ...], config_keys: list[CaseKey]) -> CaseKey | None:`

"""为给定的输入形状选择最佳的预调优配置。

选择策略:

1. 在可用的配置中找到最接近的 `intermediate_size` (优先精确匹配)。
2. 在第一步过滤后的配置中找到最接近的 `group_size` (优先精确匹配)。
3. 对于该 `intermediate_size` 和 `group_size` 下已调优的 `num_tokens`, 选择最小的 `num_tokens` 大于等于输入的 `num_tokens`。如果输入大于所有可用的 `num_tokens`, 则回退到最大的 `num_tokens`。

"""

`if not config_keys:`

`return None`

从 args 中获取形状信息

`result, _, _, group_size, *_ = args`

`num_tokens, intermediate_size = result.shape`

`cache_key = (num_tokens, group_size, intermediate_size)`

`cached = _pick_cache.get(cache_key)`

`if cached is not None:`

`return cached`

```

# 按 intermediate_size 排序并选择最接近的
sorted_by_inter = sorted(config_keys, key=lambda k: abs(k["intermediate_size"] - intermediate_size))
best_inter_key = sorted_by_inter[0]
best_inter_val = best_inter_key["intermediate_size"]

# 过滤出与最佳 intermediate_size 完全相同的配置
candidates = [k for k in config_keys if k["intermediate_size"] == best_inter_val] if
any(k["intermediate_size"] == best_inter_key["intermediate_size"] for k in config_keys) else
[best_inter_key]

# 在候选者中按 group_size 排序并选择最接近的
sorted_by_group = sorted(candidates, key=lambda k: abs(k["group_size"] - group_size))
best_group_key = sorted_by_group[0]
best_group_val = best_group_key["group_size"]

# 过滤出与最佳 group_size 完全相同的配置
candidates = [k for k in candidates if k["group_size"] == best_group_val] if any(k["group_size"]
== best_group_key["group_size"] for k in candidates) else [best_group_key]

# 在剩余的候选者中，选择最小的 num_tokens >= 输入 num_tokens
larger_candidates = [k for k in candidates if k["num_tokens"] >= num_tokens]
if larger_candidates:
    selected = min(larger_candidates, key=lambda k: k["num_tokens"])
else:
    # 没有更大的 num_tokens，回退到最大的
    selected = max(candidates, key=lambda k: k["num_tokens"])

_pick_cache[cache_key] = selected
return selected

```

tests/kernels/helion/test_silu_and_mul_per_block_quant.py

验证内核正确性和配置选择逻辑，包含多种形状和精度测试。

```

# 生成用于测试的假输入（在 FakeTensorMode 下创建）
def _generate_fake_input(
    num_tokens: int, intermediate_size: int, group_size: int
) -> tuple[Any, ...]:
    with FakeTensorMode():
        in_dtype: torch.dtype = torch.bfloat16
        out_dtype: torch.dtype = current_platform.fp8_dtype()
        scale_dtype: torch.dtype = torch.float32
        input = torch.randn(
            num_tokens, 2 * intermediate_size, device="cuda", dtype=in_dtype
        )
        result = torch.empty(
            num_tokens, intermediate_size, device=input.device, dtype=out_dtype
        )
        scale = torch.empty(

```

```

        (num_tokens, intermediate_size // group_size),
        device=input.device,
        dtype=scale_dtype,
    )
    scale_ub = torch.mean(input).to(scale_dtype)
    args = (result, input, scale, group_size, scale_ub, False)
    return args

```

```

class TestSiluAndMulPerBlockQuantConfigPicker:
    def setup_method(self):
        # 每次测试前清空缓存，保证测试独立性
        _pick_cache.clear()

    def test_config_picker_exact_match(self):
        config_keys = [
            CaseKey({"intermediate_size": 2048, "group_size": 64, "num_tokens": 16}),
            CaseKey({"intermediate_size": 4096, "group_size": 128, "num_tokens": 16}),
        ]
        args = _generate_fake_input(16, 4096, 128)
        selected_key = pick_config(args, config_keys)
        # 应精确匹配第二个配置
        assert selected_key == CaseKey(
            {"intermediate_size": 4096, "group_size": 128, "num_tokens": 16}
        )

    def test_config_picker_closest_match(self):
        # 提供多种形状，但输入的中间尺寸、分组和 token 数不在列表中
        # 应选 closest 组合
        config_keys = [
            CaseKey({"intermediate_size": 2048, "group_size": 64, "num_tokens": 16}),
            # ... 更多配置
        ]
        args = _generate_fake_input(20, 3000, 70)
        selected_key = pick_config(args, config_keys)
        # 预期选中 intermediate_size=2048, group_size=64, num_tokens=32
        assert selected_key == CaseKey(
            {"intermediate_size": 2048, "group_size": 64, "num_tokens": 32}
        )

```

评论区精华

- @yushangdi 询问 B200 配置为何缺少 H100 中的 16384 tokens 配置。作者回应：“not really. was targeting more shapes at the beginning but later reduced the scope. most of time working on h100, so there are some extra configs there.”
- @yushangdi 指出 is_scale_transposed 参数虽被参数化但从未测试。作者承认并修复。
- @yushangdi 建议使用更好的 scale_ub 生成策略（基于组间中位数而非均值），作者采纳。

- @yushangdi 建议添加注释说明内核依赖传入 scales 张量的 stride。作者回复已移除内核中的相关断言，使 `is_scale_transposed` 成为 dummy 以简化接口。
- H100 与 B200 配置差异 (question): 作者解释主要工作在 H100 上进行，因此 B200 配置范围较小。
- `is_scale_transposed` 参数未测试 (testing): 作者承认并修复。
- `scale_ub` 生成策略 (correctness): 作者采纳建议。
- 内核接口设计与 dummy 参数 (design): 决定保留 dummy 参数以简化集成。

风险与影响

- 风险：依赖 Helion 运行时（非标准 pip 包），仅支持 NVIDIA H100/B200 GPU，限制可移植性。预调优配置仅覆盖部分 `intermediate_size/group_size/num_tokens` 组合，非匹配形状可能通过最近匹配选择非最优配置。内核未集成到模型前向中，端到端收益待验证。测试使用随机正态分布输入，未覆盖边界值或极端量化场景。
- 影响：用户：需安装 Helion 包 (`pip install vllm[helion]`) 才可使用；内核默认不启用，需显式调用。系统：注册新 CustomOp，增加约 4MB 配置数据；自动调优配置选择带来轻微运行时开销（缓存命中后为常数）。团队：建立 Helion 内核集成模式 (`register_kernel + CaseKey + 配置选择`)，为后续更多内核提供参考。
- 风险标记：依赖 Helion 运行时，仅支持 H100/B200，配置覆盖不全，未集成到模型

关联脉络

- PR #32962 [Performance]: Custom Helion Kernels: 本 PR 是该 issue 的子任务，实现其中的 `silu_and_mul_per_block_quant` 内核。