PR #48512 完整报告

vllm-project/vllm

[Kernel][Helion] Add Helion kernel benchmark script

合并时间：2026-07-15 23:43

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/48512

执行摘要

- 一句话：新增 Helion 内核基准测试脚本与惰性注册
- 推荐动作：本 PR 值得精读，尤其是 benchmark_helion_kernels.py 的设计：如何选择基准测试方法（triton.do_bench vs tritonbench 的 cudagraph 实现）、如何组织基线映射。惰性注册的模式也值得在类似场景复用。风险较低，建议合入后通知相关开发者检查自定义脚本是否需要显式调用 import_all_kernels()。

功能与动机

为便于对 Helion 内核进行性能评估与调优，需要一套标准的基准测试工具。原有 baseline 依赖 CUDA ops，在自动调优和测试中不够稳定；改用原生 torch 实现可提高可重复性和易维护性。关联 Issue #32219 和 #32962 分别定义了 Helion 集成框架和候选内核列表。

实现拆解

1. 新增基准测试脚本 scripts/benchmark_helion_kernels.py（+457 行）：
   - 提供 --list 列出可用内核并提示 CUDA 基线映射是否完整。
   - 通过 --baseline 选择 autotune（默认，使用 torch.compile 包装的 autotune_baseline_fn）或 cuda（映射至 torch.ops._C.*）。
   - 支持 --no-cudagraph 禁用 CUDA graph，并在非 graph 模式下使用 triton.testing.do_bench，在 graph 模式下使用从 tritonbench 复制的 _do_bench_cudagraph_with_cache_clear（避免 L2 缓存不清理导致的性能虚高）。
   - 结果可输出至 JSON 文件。
2. 实现惰性注册：修改 vllm/kernels/helion/ops/__init__.py（+27/-5）：
   - 移除 pkgutil 自动遍历导入的逻辑，改为定义 import_all_kernels() 函数，由工具脚本显式调用以按需注册所有内核。
   - 同时更新 vllm/kernels/helion/__init__.py，删除副作用导入 import vllm.kernels.helion.ops，避免运行时自动触发注册。
3. 更新自动调优脚本 scripts/autotune_helion_kernels.py（+4 行）：
   - 在 main() 入口调用 import_all_kernels()，确保所有内核注册后再执行自动调优。
4. 为多个 Helion 内核替换 baseline 实现（涉及 silu_and_mul_per_block_quant.py、rms_norm_per_block_quant.py、rms_norm_dynamic_per_token_quant.py、

per_token_group_fp8_quant.py、dynamic_per_token_scaled_fp8_quant.py）：

- 将原 baseline() 从调用 torch.ops._C.* 改为完整的原生 PyTorch 前向计算，包括数据类型判断（int8/fp8）、残差融合、量化缩放等操作，并正确写出到输出 tensor。
- 此改动使 baseline 可作为自动调优的参考和单元测试的预期输出。

5. 删除自动导入副作用后，现有正确性测试需 rerun 以确认无回归（PR 说明已通过）。

关键文件：

- scripts/benchmark_helion_kernels.py（模块 基准测试；类别 source；类型 dependency-wiring；符号 Row, print_table, fmt, list_kernels）：新增基准测试脚手架，是 PR 的核心交付物。定义了基线映射、参数解析、性能收集与输出逻辑。
- vllm/kernels/helion/ops/__init__.py（模块 内核注册；类别 infra；类型 infrastructure；符号 import_all_kernels）：实现惰性注册机制，从自动导入改为显式函数调用，是此 PR 设计决策的关键文件。
- scripts/autotune_helion_kernels.py（模块 自动调优；类别 source；类型 dependency-wiring）：显式调用 import_all_kernels() 以适应惰性注册的变更，是配套修改的关键入口。
- vllm/kernels/helion/ops/silu_and_mul_per_block_quant.py（模块 Helion 内核；类别 infra；类型 infrastructure）：baseline() 从 CUDA op 替换为原生 PyTorch 实现，是 baseline 迁移的代表性文件。
- vllm/kernels/helion/ops/rms_norm_per_block_quant.py（模块 Helion 内核；类别 infra；类型 infrastructure）：baseline() 替换，新增残差融合和 per-block 量化逻辑。
- vllm/kernels/helion/ops/rms_norm_dynamic_per_token_quant.py（模块 Helion 内核；类别 infra；类型 infrastructure）：baseline() 替换，新增 per-token 量化实现。

关键符号：Row, print_table, list_kernels, check_requirements, make_cuda_baseline, make_autotune_baseline, cleanup_gpu_resources, import_all_kernels

关键源码片段

scripts/benchmark_helion_kernels.py

新增基准测试脚手架，是 PR 的核心交付物。定义了基线映射、参数解析、性能收集与输出逻辑。

```
# scripts/benchmark_helion_kernels.py（片段）

# 将 Helion 内核名称映射到 torch.ops._C 中的 CUDA op
# 当添加新内核且需要 CUDA 基线时，在此添加条目
CUDA_BASELINE_OPS: dict[str, str] = {
    "dynamic_per_token_scaled_fp8_quant": "dynamic_per_token_scaled_fp8_quant",
    "fused_qk_norm_rope": "fused_qk_norm_rope",
    "per_token_group_fp8_quant": "per_token_group_fp8_quant",
    "rms_norm_dynamic_per_token_quant": "rms_norm_dynamic_per_token_quant",
    "rms_norm_per_block_quant": "rms_norm_per_block_quant",
    "silu_and_mul_per_block_quant": "silu_and_mul_per_block_quant",
    "scaled_mm": "cutlass_scaled_mm",
```

```
}

# torch.compile 选项，与 vLLM 内部编译方式一致
_TORCH_COMPILE_OPTIONS: dict[str, bool] = {
    "enable_auto_functionalized_v2": False,
    "size_asserts": False,
    "alignment_asserts": False,
    "scalar_asserts": False,
    "combo_kernels": True,
    "benchmark_combo_kernel": True,
}


def make_cuda_baseline(kernel_name: str) -> Callable | None:
    """返回调用 CUDA op 的包装函数，若未映射则返回 None。"""
    if kernel_name not in CUDA_BASELINE_OPS:
        return None
    cuda_op_name = CUDA_BASELINE_OPS[kernel_name]
    cuda_op = getattr(torch.ops._C, cuda_op_name, None)
    if cuda_op is None:
        logger.warning("CUDA op %s not found", cuda_op_name)
        return None
    def baseline_fn(*args, **kwargs):
        return cuda_op(*args, **kwargs)
    return baseline_fn


def make_autotune_baseline(kernel) -> Callable | None:
    """返回 autotune_baseline_fn 经过 torch.compile 包装的函数。"""
    baseline = getattr(kernel.helion_settings, "autotune_baseline_fn", None)
    if baseline is None:
        return None
    return torch.compile(baseline, **_TORCH_COMPILE_OPTIONS)
```

vllm/kernels/helion/ops/__init__.py

实现惰性注册机制，从自动导入改为显式函数调用，是此 PR 设计决策的关键文件。

```
# vllm/kernels/helion/ops/__init__.py
"""
Helion 内核实现包。

导入此包**不会**自动注册内核。运行时代码应导入所需的特定 op 模块:
    from vllm.kernels.helion.ops import scaled_mm  # noqa: F401

需要完整注册表的工具（如 scripts/autotune_helion_kernels.py）
应显式调用 ``import_all_kernels()``。
"""

import importlib
```

```
import pkgutil


def import_all_kernels() -> list[str]:
    """导入所有内核子模块，触发每个模块的 @register_kernel 装饰器。

    Returns:
        已导入的完整模块名列表。
    """
    imported: list[str] = []
    for module_info in pkgutil.iter_modules(__path__):
        if module_info.ispkg:
            continue
        module_name = f"{__name__}.{module_info.name}"
        importlib.import_module(module_name)
        imported.append(module_name)
    return imported
```

vllm/kernels/helion/ops/silu_and_mul_per_block_quant.py

baseline() 从 CUDA op 替换为原生 PyTorch 实现，是 baseline 迁移的代表性文件。

```
# vllm/kernels/helion/ops/silu_and_mul_per_block_quant.py 中的 baseline 片段

def baseline(
    out: torch.Tensor,
    input: torch.Tensor,
    scales: torch.Tensor,
    group_size: int,
    scale_ub: torch.Tensor | None = None,
    is_scale_transposed: bool = False,
) -> None:
    # 改用原生 PyTorch 实现，不再调用 torch.ops._C.silu_and_mul_per_block_quant
    num_tokens, intermediate_size = out.shape
    groups_per_row = intermediate_size // group_size
    quant_dtype = out.dtype

    if quant_dtype == torch.int8:
        qtype_min, qtype_max = get_int8_min_max()
        min_scaling_factor = get_int8_min_scaling_factor()
    else: # fp8
        qtype_min, qtype_max = get_fp8_min_max()
        min_scaling_factor = 1.0 / (qtype_max * 512.0)

    # SiLU 激活 → 分组求绝对最大值 → 计算 scale → 量化
    act = SiluAndMul.forward_native(input.to(torch.float32))
    x_grouped = act.view(num_tokens, groups_per_row, group_size)
    s = torch.amax(torch.abs(x_grouped), dim=-1)
    if scale_ub is not None:
        s = s.clamp(max=scale_ub)
```

```
    s = (s * (1.0 / qtype_max)).clamp(min=min_scaling_factor)

    y = x_grouped / s[:, :, None]
    if quant_dtype == torch.int8:
        y = y.round()

    scales.copy_(s)
    out.copy_(
        y.clamp(qtype_min, qtype_max).view(num_tokens, intermediate_size).to(out.dtype)
    )
```

评论区精华

无实质讨论。yushangdi 和 zou3519 直接批准，claude[bot] 自动评论无实际内容。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 惰性注册破坏现有代码：若其他模块依赖 import vllm.kernels.helion.ops 时的自动注册，则会因注册失效而崩溃。但该 PR 同时更新了 autotune_helion_kernels.py 显式调用 import_all_kernels()；需要确保所有需注册的场景都做了相应调整。
  2. baseline 替换引入数值误差：原生 PyTorch 实现与 CUDA op 可能存在极小的数值差异，可能影响自动调优的配置选择或单元测试的容差判断。PR 声称已通过正确性测试，但未提供覆盖边界 case（如极端值、inf/nan）的证明。
  3. 新基准测试脚本本身无单元测试：脚本逻辑（如参数解析、基线函数分发、性能数据收集）未经自动化测试，后续修改可能导致功能退化。
  4. CUDA graph 模式依赖 tritonbench 复制代码：复制自外部仓库的函数缺少版本跟踪，若 tritonbench 有 bug 修复将不会自动同步。 - 影响：开发者：获得统一的 Helion 内核性能评估工具，可标准化未来内核优化的验证流程。惰性注册减少了非 Helion 场景的无关导入开销。系统：运行时行为无变化，因基准测试脚本仅作为独立工具使用。自动调优和测试流程依赖新的注册方式，需注意调用顺序。团队：内核开发流程更加清晰（编写内核 → 注册 → 基准测试 → 自动调优）。需要维护 CUDA_BASELINE_OPS 映射以确保基线完整性。
- 风险标记：惰性注册可能破坏依赖自动导入的代码，baseline 替换存在数值精度风险，基准测试脚本无单元测试覆盖

关联脉络

- 暂无明显关联 PR