PR #25274 完整报告

sgl-project/sglang

[Refactor] JIT kernel benchmark

合并时间：2026-05-28 15:49

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25274

执行摘要

- 一句话：重写 JIT kernel benchmark 框架，替换 triton.testing
- 推荐动作：建议精读此 PR，尤其是 marker.py 中 do_bench 的实现和 parametrize 的 pytest 风格设计。它为 CUDA kernel benchmark 提供了一套可复用的轻量方案，值得其他项目借鉴。bench_qknorm.py 的迁移展示了如何大幅简化代码。

功能与动机

PR body 指出 triton.testing.benchmark 过于沉重且存在 bug：1）缺失当前 CUDA stream 的同步；2）无法在 CUDA graph benchmark 中冷 L2 缓存。以往通过在代码中手动模拟多层来规避 L2 缓存效果，导致代码混乱难以理解。因此需要一个轻量级 benchmark 工具，提升可读性和结果准确性。

实现拆解

1. 创建核心 benchmark 框架：新增 python/sglang/jit_kernel/benchmark/marker.py，定义 Benchmark 类、parametrize 装饰器、do_bench 函数等核心组件。支持 CUDA graph 捕获、自动流同步、buffer 克隆以冷 L2、内存带宽计算。
2. 重写示例 benchmark：修改 bench_qknorm.py 和 bench_store_cache.py，用 @marker.parametrize 声明参数网格，@marker.benchmark 声明比较轴，替代原来的 itertools.product 和 triton.testing.Benchmark。移除手动多层模拟代码，改为框架内部通过 graph_clone_args 刷新缓存。
3. 适配更多 benchmark：更新 bench_activation.py 同样使用新框架，简化了 activation 和 filter benchmark 的编写。
4. 增加工具函数：在 utils.py 中添加 create_empty 和 create_random 便捷函数，统一使用默认 dtype/device。
5. 更新开发文档：修改 .claude/skills/add-jit-kernel/SKILL.md，详细说明 marker 框架用法，包括参数化、do_bench 关键参数和示例代码。

关键文件：

- python/sglang/jit_kernel/benchmark/marker.py（模块 基准框架；类别 source；类型 core-logic；符号 BenchSkip, skip, _get_benchmark_stream, _clone_recursive）：核心新增文件，定义了整个 benchmark 框架，包括 Benchmark 类、parametrize 装饰器、do_bench 函数、BenchResult 和 Table 等，是 PR 的核心贡献。

- python/sglang/jit_kernel/benchmark/bench_qknorm.py（模块 基准测试；类别 source；类型 core-logic；符号 sglang_jit_qknorm, flashinfer_qknorm, benchmark）：示例迁移，展示如何用新框架重写 benchmark，去掉大量样板代码，变为声明式参数化。
- python/sglang/jit_kernel/benchmark/bench_store_cache.py（模块 基准测试；类别 source；类型 core-logic；符号 sglang_jit_store_cache, benchmark, fn）：第二个迁移示例，验证新框架对多参数和 CUDA graph 的支持。
- python/sglang/jit_kernel/benchmark/bench_activation.py（模块 基准测试；类别 source；类型 core-logic；符号 benchmark, f）：第三个迁移示例，涵盖 activation 和 filter 两个 benchmark。
- python/sglang/jit_kernel/benchmark/utils.py（模块 工具函数；类别 source；类型 core-logic；符号 create_empty, create_random）：新增 create_empty/create_random 辅助函数，简化张量创建。
- .claude/skills/add-jit-kernel/SKILL.md（模块 开发者文档；类别 docs；类型 documentation；符号 torch_impl_scale, benchmark）：更新开发文档，详细介绍 marker 框架的用法，帮助团队快速上手。

关键符号：skip, do_bench, parametrize, benchmark, Benchmark.run, create_random, create_empty, sglang_aot_qknorm, torch_impl_qknorm

关键源码片段

python/sglang/jit_kernel/benchmark/marker.py

核心新增文件，定义了整个 benchmark 框架，包括 Benchmark 类、parametrize 装饰器、do_bench 函数、BenchResult 和 Table 等，是 PR 的核心贡献。

```
# marker.py - 轻量级 JIT kernel benchmark 框架核心

import torch
from sglang.jit_kernel.utils import cache_once

# 缓存每个设备的 CUDA stream，用于确保 benchmark 在独立 stream 上进行 @cache_once
def _get_benchmark_stream(device_id: int) -> torch.cuda.Stream:
    return torch.cuda.Stream(device=device_id)

# 递归克隆输入，用于每次 graph 重放时刷新 L2 缓存
def _clone_recursive(in_: Any) -> Any:
    if isinstance(in_, torch.Tensor):
        return in_.clone()
    elif isinstance(in_, (list, tuple)):
        return type(in_)(_clone_recursive(x) for x in in_)
    elif isinstance(in_, dict):
        return {k: _clone_recursive(v) for k, v in in_.items()}
    # 基本类型直接返回
    elif isinstance(in_, (bool, int, float, str, torch.dtype, torch.device, type(None))):
        return in_
    raise ValueError(f"unsupported type: {type(in_)}")
```

```
# 递归获取张量总字节数（用于带宽估算）
def _get_nbytes_recursive(in_: Any) -> int:
    if isinstance(in_, torch.Tensor):
        return in_.nbytes
    elif isinstance(in_, (list, tuple)):
        return sum(_get_nbytes_recursive(x) for x in in_)
    elif isinstance(in_, dict):
        return sum(_get_nbytes_recursive(v) for v in in_.values())
    elif isinstance(in_, (bool, int, float, str, torch.dtype, torch.device, type(None))):
        return 0
    raise ValueError(f"unsupported type: {type(in_)}")

def _process_metrics(times: list[float], metrics: tuple[Metric, ...]) -> list[float]:
    # 排序后将微秒转换为秒
    times = sorted(x / 1000 for x in times)
    results = []
    for metric in metrics:
        if metric == "avg":
            results.append(sum(times) / len(times))
        else:
            # 取指定分位数
            which = min(int(len(times) * metric), len(times) - 1)
            results.append(times[which])
    return results

class BenchResult(NamedTuple):
    metrics: Tuple[Metric, ...]
    times: List[float] # 单位：秒
    memory_footprint: Optional[int]

class Table:
    # 对齐文本表格，用于打印结果
    SEP = " | "
    def __init__(self):
        self._headers: List[str] = []
        self._mins: List[int] = []
        self._pads: List[int] = []
        self._aligns: List[str] = []
        self._seps: set = set()
        self._rows: List[List[str]] = []

    @staticmethod
    def format_latency(r: float) -> str:
        if math.isnan(r):
            return "N/A"
        length = len(str(int(r)))
        if length < 5:
            return f"{r:.4f}"
        digits = max(0, 4 - (length - 5))
```

```
        return f"{r:.{digits}f}"

    @staticmethod
    def format_bandwidth(b: float) -> str:
        if math.isnan(b):
            return "N/A"
        return f"{b:.2f}"
    # ... (其余方法用于构建和打印表格)
```

python/sglang/jit_kernel/benchmark/bench_qknorm.py

示例迁移，展示如何用新框架重写 benchmark，去掉大量样板代码，变为声明式参数化。

```
# bench_qknorm.py - 使用新框架的示例 benchmark

import torch
from sglang.jit_kernel.benchmark import marker
from sglang.jit_kernel.benchmark.utils import create_random
from sglang.jit_kernel.norm import fused_inplace_qknorm
from sglang.srt.utils import get_current_device_stream_fast

FN_MAP = {
    "aot": sglang_aot_qknorm, # 改用 flashinfer rmsnorm
    "jit": fused_inplace_qknorm,
    "torch": torch_impl_qknorm,
}

# @marker.parametrize 声明参数网格（每个参数可独立指定 full 和 CI 下的值）
# @marker.benchmark 声明比较轴 line_arg="impl"，取值为 ["aot", "jit", "torch"]
@marker.parametrize("head_dim", [128, 256, 512, 1024], [128])
@marker.parametrize("GQA", [4, 8], [4])
@marker.parametrize("num_kv_heads", [1, 2, 4, 8], [1])
@marker.parametrize("batch_size", [2**n for n in range(0, 14)], [16])
@marker.benchmark("impl", ["aot", "jit", "torch"])
def benchmark(head_dim: int, GQA: int, num_kv_heads: int, batch_size: int, impl: str):
    num_qo_heads = GQA * num_kv_heads
    q = create_random(batch_size, num_qo_heads, head_dim)
    k = create_random(batch_size, num_kv_heads, head_dim)
    q_weight = create_random(head_dim)
    k_weight = create_random(head_dim)
    # do_bench 负责 CUDA graph 捕获、L2 缓存规避、内存带宽计算
    return marker.do_bench(
        FN_MAP[impl],
        input_args=(q, k, q_weight, k_weight),
        memory_output=(q, k), # 标记 inplace 写出的张量
    )

if __name__ == "__main__":
    benchmark.run()
```

评论区精华

Review 中 gemini-code-assist 提出了多项建议：

- 带宽计算单位歧义：内部计算使用 GiB 但输出标签为 GB/s，建议统一为 GiB/s 或修改计算。
- IndexError 风险：当 line_vals 为空时，需要对空列表增加守卫。
- memory_args 计数：in-place kernel（如 qknorm）应重复传入读写张量以准确计量带宽，建议 memory_args=(q, q, k, k, q_weight, k_weight)。
- 索引唯一性：store cache benchmark 应使用 torch.randperm 避免重复索引导致的写冲突（但原代码已使用，建议多余）。
- graph_clone_args 扩展：需要克隆所有缓存 buffer 才能完全规避 L2 缓存效果，建议将 k_cache、v_cache 也加入克隆列表。
- memory_args 包含写操作：store cache benchmark 应计入 k 和 v 的写入，建议 memory_args=(k, k, v, v, indices)。

这些讨论中，部分已在 head 代码中实现或原本就正确（如 randperm），部分未采纳（如 memory_args 重复计数）。

- 带宽计算单位歧义（GiB vs GB/s）(design): 未在提交中显式修复，可能维持现状未采纳。
- memory_args 对 in-place kernel 应双倍计数 (correctness): head 代码中 memory_args 仍为默认（memory_args="all" 实际仅计一次），未采纳建议。
- store cache 索引唯一性 (performance): 原代码和 head 代码均已使用 torch.randperm，建议多余。

风险与影响

- 风险：
  1. 框架稳定性：新框架 marker.py 仅应用于少量 benchmark，可能存在未发现的 CUDA graph 兼容性或数值问题。
  2. CI 基准波动：由于加入了 L2 缓存规避和流同步，benchmark 结果可能更稳定但也可能因 GPU 型号差异产生新的波动。
  3. 内存开销：CUDA graph 克隆 buffer 会增加显存占用，尤其在大型 tensor 场景下可能 OOM。
  4. 单位歧义：带宽计算使用 GiB 但标签为 GB/s，可能造成数据误读。
  5. 迁移不完整：部分旧 benchmark（如 bench_activation.py 中 filter 部分）已迁移，但仍有其他未迁移的 benchmark，维护两个模式成本。 - 影响：用户：JIT kernel 开发者编写 benchmark 的体验显著改善——使用声明式参数化、自动 CUDA graph、准确带宽计算。系统：benchmark 结果对 L2 缓存影响更鲁棒，stream 同步使计时准确。团队：需要推广新框架，逐步迁移所有旧 benchmark 并废弃 triton.testing 用法。文档 SKILL.md 已更新，降低学习门槛。 - 风险标记：新框架稳定性风险，单位混淆风险，CI 基准波动可能

关联脉络

- 暂无明显关联 PR