PR #23745 完整报告

sgl-project/sglang

Use Cute-DSL NVFP4 quantization kernels

合并时间：2026-05-11 15:40

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23745

执行摘要

- 一句话：SM100 默认使用 Cute-DSL NVFP4 量化，性能提升
- 推荐动作：本 PR 值得关注其通过注册 custom_op 实现 CUDA graph 兼容的技巧，以及在不同后端间自动选择的设计模式。对于要修改量化后端的开发者，是很好的参考。

功能与动机

在 Blackwell（SM100）上，FlashInfer 的 Cute-DSL FP4 量化核经过性能优化后全面超越原始 CUDA 核（参见 flashinfer#2904），因此需要将其集成并设为默认后端，以获得最高性能。

实现拆解

1. 在 fp4_utils.py 中引入 FlashInfer 的 fp4_quantize，根据 SM100 判断自动选择 cute-dsl 或 cuda backend。
2. 使用 register_custom_op_from_extern 注册量化算子，并提供 fake 实现以支持 CUDA graph 捕获。
3. 移除 modelopt_quant.py 中旧的 fp4_quantize 导入和 fallback 代码，改为从 fp4_utils 导入。
4. 更新 standard.py、flashinfer.py、compressed_tensors 等文件中的导入路径，统一引用 fp4_utils 的 fp4_quantize。
5. 重写 bench_fp4_quant.py 基准脚本，直接对比 sglang 和 flashinfer 的量化性能，并增加绘图和 CSV 输出。

关键文件：

- benchmark/kernels/quantization/bench_fp4_quant.py（模块 性能测试；类别 source；类型 benchmark；符号 _bench, benchmark, main, plot_speedup）：完全重写，用于对比 sglang jit kernel 与 FlashInfer Cute-DSL 的 FP4 量化性能，新增绘图函数和 CSV 输出，验证新后端性能优势。
- python/sglang/srt/layers/quantization/fp4_utils.py（模块 量化工具；类别 source；类型 core-logic；符号 _round_up, _flashinfer_fp4_quantize_impl, _flashinfer_fp4_quantize_fake）：核心变更文件，集中了 FP4 量化函数的包装逻辑，通过 register_custom_op_from_extern 注册可被 CUDA graph 捕获的算子，并根据 SM100 自动选择 cute-dsl 后端。

- python/sglang/srt/layers/quantization/modelopt_quant.py（模块 量化配置；类别 source；类型 dependency-wiring）：删除旧有的 fp4_quantize 导入逻辑，改为从 fp4_utils 导入，清理代码并统一入口。

关键符号：_flashinfer_fp4_quantize_impl, _flashinfer_fp4_quantize_fake, _round_up, benchmark, _bench, main, plot_speedup

关键源码片段

benchmark/kernels/quantization/bench_fp4_quant.py

完全重写，用于对比 sglang jit kernel 与 FlashInfer Cute-DSL 的 FP4 量化性能，新增绘图函数和 CSV 输出，验证新后端性能优势。

```
"""Benchmark FP4 quantize: sglang jit_kernel vs flashinfer.

Compares ``sglang.jit_kernel.nvfp4.scaled_fp4_quant`` against
``flashinfer.fp4_quantize`` over a sweep of (M, K) shapes.

Timing uses ``flashinfer.testing.bench_gpu_time`` (CUDA-graph based with
rotating-buffer cold-L2).
"""

import argparse
import itertools

import numpy as np
import torch
from flashinfer import fp4_quantize as flashinfer_fp4_quantize
from flashinfer.testing import bench_gpu_time

from sglang.jit_kernel.nvfp4 import scaled_fp4_quant

Ms = [1, 8, 32, 128, 512, 1024, 2048, 4096, 8192, 16384, 32768]
Ks = [128, 256, 384, 512, 768, 1024, 1536, 2048, 3072, 4096, 5120, 6144, 8192, 16384]


def _bench(fn, input_args) -> float:
    """用 flashinfer 的 bench_gpu_time 进行 CUDA-graph 计时，返回中位数耗时（毫秒）。"""
    times = bench_gpu_time(
        fn=fn,
        input_args=input_args,
        use_cuda_graph=True,
        dry_run_time_ms=25,
        repeat_time_ms=100,
    )
    return float(np.median(times))


def benchmark(M: int, K: int, dtype: torch.dtype, device: str):
```

```
    """对给定形状 (M, K) 分别运行 sglang 和 flashinfer 的量化，返回各自耗时（毫秒）。"""
    x = torch.randn(M, K, device=device, dtype=dtype)
    global_scale = torch.ones(1, device=device, dtype=torch.float32)

    sglang_ms = _bench(
        lambda x, gs: scaled_fp4_quant(x, gs),
        input_args=(x, global_scale),
    )
    flashinfer_ms = _bench(
        lambda x, gs: flashinfer_fp4_quantize(x, gs, backend="cute-dsl"),
        input_args=(x, global_scale),
    )
    return sglang_ms, flashinfer_ms


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--dtype", choices=["bf16", "fp16"], default="bf16")
    parser.add_argument("--device", default="cuda")
    parser.add_argument("--csv", type=str, default=None)
    parser.add_argument("--plot", type=str, default=None)
    args = parser.parse_args()

    dtype = torch.bfloat16 if args.dtype == "bf16" else torch.float16

    rows = []
    header = f"{'M':>8} {'K':>8} {'sglang(us)':>12} {'flashinfer(us)':>16} {'speedup':>10}"
    print(header)
    print("-" * len(header))

    for M, K in itertools.product(Ms, Ks):
        try:
            sglang_ms, flashinfer_ms = benchmark(M, K, dtype, args.device)
        except Exception as e:
            print(f"{M:>8} {K:>8}  skipped: {e}")
            continue
        # 转换为微秒
        sglang_us = sglang_ms * 1e3
        flashinfer_us = flashinfer_ms * 1e3
        speedup = flashinfer_us / sglang_us # >1 表示 flashinfer 更快
        print(f"{M:>8} {K:>8} {sglang_us:>12.3f} {flashinfer_us:>16.3f} {speedup:>10.3f}")
        rows.append((M, K, sglang_us, flashinfer_us, speedup))

    if args.csv:
        with open(args.csv, "w") as f:
            f.write("M,K,sglang_us,flashinfer_us,speedup_flashinfer_over_sglang\n")
            for M, K, s, fi, sp in rows:
                f.write(f"{M},{K},{s:.6f},{fi:.6f},{sp:.6f}\n")
        print(f"Saved CSV to {args.csv}")
```

```
    if args.plot:
        plot_speedup(rows, args.plot)


if __name__ == "__main__":
    main()
```

评论区精华

Fridge003 建议将 fp4_quantize 从 modelopt_quant.py 移到 fp4_utils.py，作者立即执行。对于是否删除未使用的 jit kernel，作者表示可作为后续改进。关于 piecewise CUDA graph 兼容性，作者展示了启动服务器的成功日志，证明 register_custom_op 方案可以正常工作。

- fp4_quantize 函数位置重构 (design): b8zhong 接受建议并移动。
- 未使用的 jit kernel 是否删除 (style): b8zhong 表示可作为后续工作，暂不删除。
- piecewise CUDA graph 兼容性测试 (correctness): b8zhong 展示了启动服务器并使用 CUDA graph 的成功日志，证明 register_custom_op 方案有效。

风险与影响

- 风险：依赖 FlashInfer 新版本（需支持 cute-dsl backend），若 flashinfer 不可用则 fp4_quantize 为 None，需注意 fallback。默认 backend 改为 cute-dsl，在非 SM100 设备上自动回退到 cuda，但可能因 flashinfer 版本差异导致行为不一致。旧代码中依赖 sglang jit kernel 的路径虽未被删除，但已不再使用，若有外部代码直接引用可能失效。
- 影响：用户：在搭载 Blackwell GPU 的系统上，FP4 量化推理速度提升明显；开发者：量化函数集中管理，后续维护更容易；测试：基准测试脚本重写，支持更全面的 shape 覆盖和绘图。
- 风险标记：依赖 FlashInfer 版本，CUDA Graph 兼容依赖 custom_op，移除旧导入对外部代码的影响

关联脉络

- 暂无明显关联 PR