PR #31340 完整报告

sgl-project/sglang

Fix FP8 Triton dtype selection on A100

合并时间：2026-07-25 08:03

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/31340

执行摘要

- 一句话：修复 A100 上 FP8 Triton dtype 选择
- 推荐动作：建议所有涉及 FP8 量化的开发者仔细审阅此 PR，特别是使用 A100 GPU 的用户。值得关注的设计决策包括：基于 CUDA capability 的 Triton dtype 抽象（fp8_dtype_to_triton）和通过 uint8 视图规避 FP8 指针限制的技巧。推荐在合并前针对 A100 添加显式的测试用例，并确认 Marlin fallback 在 pre-Hopper GPU 上正常工作。

功能与动机

在 A100/SM80 GPU 上，Triton 将 E4M3 FP8 命名为 tl.float8e4b15，而不是 Blackwell 上使用的 tl.float8e4nv。直接使用 tl.float8e4nv 会导致内核崩溃。此外，A100 不支持 FP8 指针类型，需要将 FP8 值通过 uint8 视图进行位转换再存储。ModelOpt FP8 线性路径在 pre-Hopper GPU 上缺少 Marlin fallback，导致无法在这些显卡上使用。

实现拆解

1. 新增工具函数（fp8_utils.py）：引入 cuda_capability_uses_fp8_e4b15、use_fp8_e4b15_for_e4m3fn 和 fp8_dtype_to_triton，根据 CUDA capability 统一选择正确的 Triton dtype，并为非 CUDA 平台提供安全 fallback。
2. 修改量化内核（fp8_kernel.py、fp8_quantize.py）：在 _static_quant_fp8、_per_tensor_quant_mla_fp8_stage2、_per_token_group_quant_mla_deep_gemm_masked_fp8、_fp8_quantize_kernel 等内核中，将显式的 tl.float8e4nv 替换为 FP8_DTYPE 编译时常量（来自 fp8_dtype_to_triton），并将输出 store 改为 tl.store(..., x_fp8.to(tl.uint8, bitcast=True), ...)，同时将输入指针转换为 uint8 视图。
3. 启用 ModelOpt FP8 Marlin fallback（modelopt_quant.py）：在 ModelOptFp8LinearMethod.__init__ 中，判断 CUDA 环境后通过 can_auto_enable_marlin_fp8() 或环境变量 SGLANG_FORCE_FP8_MARLIN 自动启用 Marlin fallback；在 process_weights_after_loading 中调用 prepare_fp8_layer_for_marlin 并删除无用属性 input_scale；在 apply 方法中分发到 Marlin 线性算子。

关键文件：

- python/sglang/kernels/ops/quantization/fp8_utils.py（模块 量化内核；类别 infra；类型 infrastructure；符号 cuda_capability_uses_fp8_e4b15, use_fp8_e4b15_for_e4m3fn,

fp8_dtype_to_triton）：新增核心工具函数，统一 Triton FP8 dtype 选择逻辑

- python/sglang/srt/layers/quantization/modelopt_quant.py（模块 量化配置；类别 source；类型 data-contract）：添加 Marlin fallback 支持，扩展 ModelOpt FP8 的硬件兼容性
- python/sglang/kernels/ops/quantization/fp8_kernel.py（模块 量化内核；类别 infra；类型 infrastructure；符号 _static_quant_fp8, _per_tensor_quant_mla_fp8_stage2, _per_token_group_quant_mla_deep_gemm_masked_fp8）：修改静态和动态量化内核，使用正确的 dtype 和 uint8 视图
- python/sglang/kernels/ops/quantization/fp8_quantize.py（模块 量化内核；类别 infra；类型 infrastructure；符号 _fp8_quantize_kernel, fp8_quantize）：修改 fp8_quantize 核函数，兼容不同 Triton dtype

关键符号：cuda_capability_uses_fp8_e4b15, use_fp8_e4b15_for_e4m3fn, fp8_dtype_to_triton, _static_quant_fp8, _per_tensor_quant_mla_fp8_stage2, _per_token_group_quant_mla_deep_gemm_masked_fp8, _fp8_quantize_kernel, ModelOptFp8LinearMethod.init, ModelOptFp8LinearMethod.process_weights_after_loading, ModelOptFp8LinearMethod.apply

关键源码片段

python/sglang/kernels/ops/quantization/fp8_utils.py

新增核心工具函数，统一 Triton FP8 dtype 选择逻辑

```
# file: python/sglang/kernels/ops/quantization/fp8_utils.py
from __future__ import annotations

from typing import Optional, Tuple

import torch
import triton.language as tl

from sglang.kernels.jit.utils import (
    get_jit_cuda_arch,
    is_hip_runtime,
    is_musa_runtime,
)

# Triton 在 SM89 之前的架构上将 E4M3 称为 fp8e4b15
def cuda_capability_uses_fp8_e4b15(cuda_capability: Tuple[int, int]) -> bool:
    return cuda_capability < (8, 9)


def use_fp8_e4b15_for_e4m3fn(
    device: Optional[int] = None,
    cuda_capability: Optional[Tuple[int, int]] = None,
) -> bool:
    """判断当前设备是否需要对 E4M3 使用 e4b15 变体。"""
    if cuda_capability is None:
```

```
        # 非 CUDA 平台统一返回 False
        if is_hip_runtime() or is_musa_runtime() or not torch.cuda.is_available():
            return False
        if device is None:
            arch = get_jit_cuda_arch()
            cuda_capability = (arch.major, arch.minor)
        else:
            cuda_capability = torch.cuda.get_device_capability(device)
    return cuda_capability_uses_fp8_e4b15(cuda_capability)


def fp8_dtype_to_triton(
    fp8_dtype: torch.dtype,
    *,
    device: Optional[int] = None,
    cuda_capability: Optional[Tuple[int, int]] = None,
) -> tl.dtype:
    """将 PyTorch FP8 dtype 映射为 Triton 支持的 dtype 常量。
    根据设备能力选择 e4b15 / e4nv / e4b8 / e5 之一。
    """
    if fp8_dtype == torch.float8_e4m3fn:
        if use_fp8_e4b15_for_e4m3fn(device, cuda_capability):
            return tl.float8e4b15
        return tl.float8e4nv
    if fp8_dtype == torch.float8_e4m3fnuz:
        return tl.float8e4b8
    if fp8_dtype == torch.float8_e5m2:
        return tl.float8e5
    raise ValueError(f"Unsupported FP8 dtype: {fp8_dtype}")
```

python/sglang/srt/layers/quantization/modelopt_quant.py

添加 Marlin fallback 支持，扩展 ModelOpt FP8 的硬件兼容性

```
# file: python/sglang/srt/layers/quantization/modelopt_quant.py ( 新增部分 )
class ModelOptFp8LinearMethod(LinearMethodBase):
    def __init__(self, quant_config: ModelOptFp8Config):
        super().__init__()
        # ... 原有初始化 ...
        # 新增：Marlin fallback 检测
        self.use_marlin = False
        if is_cuda():
            self.use_marlin = (
                envs.SGLANG_FORCE_FP8_MARLIN.get() or can_auto_enable_marlin_fp8()
            )

    def process_weights_after_loading(self, layer: torch.nn.Module) -> None:
        # ... 原有 weight scale 处理 ...
        # 新增：若启用 Marlin，准备层并删除 input_scale
        if self.use_marlin:
```

```
            prepare_fp8_layer_for_marlin(layer)
            del layer.input_scale # Marlin 是 weight-only，不需要 input_scale

    def apply(self, layer, x, bias=None):
        # 新增：Marlin 快速路径
        if self.use_marlin:
            return torch.ops.sglang.apply_fp8_marlin_linear(
                input=x, weight=layer.weight,
                weight_scale=layer.weight_scale,
                workspace=layer.workspace,
                size_n=layer.output_size_per_partition,
                size_k=layer.input_size_per_partition,
                bias=bias,
            )
        # 原有 flashinfer / torch._scaled_mm 路径 ...
```

评论区精华

Review 中主要讨论了三个问题：

1. 环境变量读取方式：b8zhong 指出应使用 envs 模块代替 get_bool_env_var，danielafrimi 已修正。
2. 新增文件位置：b8zhong 建议将 dtype 选择函数合并到 fp8_utils.py 中，避免单独文件，被采纳。
3. Marlin 与 input_scale 的关系：danielafrimi 说明删除 input_scale 是因为 Marlin 是 weight-only 量化，不需要激活 scale，得到了认可。

- 使用 envs 模块代替 get_bool_env_var (style): danielafrimi 已修改为 envs.SGLANG_FORCE_FP8_MARLIN.get()
- 将 dtype 选择函数移到 fp8_utils.py (design): 作者采纳建议，将函数移入 fp8_utils.py，删除了独立文件
- Marlin 是 weight-only，删除 input_scale (correctness): 无争议，直接接受

风险与影响

- 风险：主要风险包括：
- 回归风险：量化内核（fp8_kernel.py、fp8_quantize.py）的修改会影响所有使用这些核函数的场景，包括动态和静态量化，需在 A100 和 H100 上做性能回归测试。
- 兼容性风险：fp8_dtype_to_triton 对非 CUDA 平台返回 False，但 HIP 和 MUSA 运行时可能仍需要调整，目前缺乏测试。
- Marlin fallback 的依赖风险：can_auto_enable_marlin_fp8() 的实现细节可能随硬件环境变化，若 Marlin 内核未正确编译会导致运行时错误。
- 缺少测试覆盖：本次改动未包含直接的单元测试或集成测试，需要补充针对 A100 的 FP8 dtype 选择和 Marlin fallback 的测试用例。
- 影响：对用户的影响：

- A100/SM80 用户将能够正常使用 FP8 E4M3 量化，不再因 Triton dtype 不匹配而崩溃。
- ModelOpt FP8 检查点用户在 pre-Hopper GPU 上可通过 Marlin fallback 运行，扩大了 FP8 量化的硬件覆盖面。

对系统的影响：所有调用 scaled_fp8_quant、static_quant_fp8、per_tensor_quant_mla_fp8 的模块（如 DeepSeek 等）均受影响，但行为语义不变。

对团队的影响：需关注内核性能测试结果，并考虑增加 A100 CI runner 以覆盖该架构。

- 风险标记：量化内核变更，兼容性风险，缺少测试覆盖

关联脉络

- PR #32296 [Perf] Halve the non-finite sanitization overhead in per_token_group_quant: 同属量化内核优化系列，但侧重点不同（本 PR 修复 dtype 正确性，32296 优化性能）
- PR #32288 Fix stale flashinfer-MLA fallback poisoning spec verify capture (trtllm_mla + tc_piecewise): 同为 fallback 机制修复，思路类似（本 PR 为 Marlin fallback，32288 为 flashinfer-MLA fallback）
- PR #31346 fix(dsa): fail fast on fp8_e4m3 KV with tilelang DSA backend on CUDA: 同为 FP8 兼容性修复，但针对不同场景（KV cache vs 权重量化）