PR #23833 完整报告

sgl-project/sglang

[JIT Kernel][1/2]Migrate MXFP8 Group GEMM & Quant into JIT

合并时间：2026-04-29 22:50

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23833

执行摘要

- 一句话：MXFP8 MoE Group GEMM & Quant 迁移至 JIT，Blackwell 加速
- 推荐动作：值得精读，尤其是 Python 端 JIT 编译集成模式（cache_once + load_jit）和 MoE Group GEMM 的 CUTLASS 实现。2SM 策略在 memory-bound 场景的加速效果值得关注。作为系列 PR 的第 1 部分，建议跟踪后续集成。

功能与动机

现有的 sgl-kernel 中的 MXFP8 操作是预编译的，缺乏灵活性。迁移到 JIT 可以利用 TVM 动态编译适配当前 GPU，并允许采用更高效的 MoE Group GEMM 替代普通 Group GEMM，减少 TMA 更新并提高性能和内存受限场景的吞吐量。

实现拆解

1. 在 python/sglang/jit_kernel/mxfp8.py 中封装 JIT 编译逻辑：定义 _mxfp8_cuda_flags() 和 _mxfp8_arch_env() 设置 NVCC 参数（启用 CUDA 12.8 sm_100a）。通过 @cache_once 的 _jit_es_sm100_mxfp8_blockscaled_group_quant 和 _jit_es_sm100_mxfp8_blockscaled_moe_group_gemm 使用 load_jit 加载 .cuh 源文件并编译为 TVM Module。对外提供两个高层函数 es_sm100_mxfp8_blockscaled_grouped_quant 和 es_sm100_mxfp8_blockscaled_moe_grouped_gemm，自动调用编译好的内核。
2. 在 python/sglang/jit_kernel/csrc/moe/expert_specialization/ 下实现 CUDA 内核：
   - es_sm100_mxfp8_blockscaled_group_quant.cuh：按 128 元素块计算 max 并导出 fp8 和 scale factor。
   - es_sm100_mxfp8_blockscaled_moe_group_gemm.cuh：使用 CUTLASS GemmUniversalAdapter，以 MoEProblemShape 驱动 grouped GEMM，先通过预计算核生成指针数组，再运行主 GEMM。
   - es_sm100_mxfp8_blockscaled_moe_group_gemm_traits.cuh：定义 2SM 和 1SM 等 MMA 配置。
   - es_sm100_mxfp8_blockscaled_moe_group_gemm_functor.cuh：偏移计算 functor。
3. 在 python/sglang/jit_kernel/tests/test_mxfp8_moe.py 添加单元测试：参数化 num_experts 和输出类型，通过 calc_diff 与 fp32 参考对比，精度要求 diff < 0.001。使用 register_cuda_ci 注册到 CI 套件。

4. 在 python/sglang/jit_kernel/benchmark/bench_mxfp8_moe.py 添加基准测试，与 sgl-kernel 版本对比延迟。
5. 通过 CI 注册确保 GPU 资源分配；lint 和格式调整。

关键文件：

- python/sglang/jit_kernel/mxfp8.py（模块 JIT 封装；类别 source；类型 dependency-wiring；符号 es_sm100_mxfp8_blockscaled_grouped_quant, es_sm100_mxfp8_blockscaled_moe_grouped_gemm, _jit_es_sm100_mxfp8_blockscaled_group_quant, _jit_es_sm100_mxfp8_blockscaled_moe_group_gemm）：JIT 封装入口，提供量化和 GEMM 的高层接口，串联所有 CUDA 内核。
- python/sglang/jit_kernel/tests/test_mxfp8_moe.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_es_sm100_mxfp8_blockscaled_grouped_mm, align, calc_diff, is_sm100_supported）：单元测试验证精度，覆盖多 expert 和输出类型。
- python/sglang/jit_kernel/benchmark/bench_mxfp8_moe.py（模块 基准测试；类别 source；类型 dependency-wiring；符号 is_sm100_supported, _probe_sgl_kernel_group_mm, _prepare_case, benchmark）：基准测试比较 JIT 和 sgl-kernel 延迟。
- python/sglang/jit_kernel/csrc/moe/expert_specialization/es_sm100_mxfp8_blockscaled_group_quant.cuh（模块 CUDA 内核；类别 other；类型 dependency-wiring）：量化的 CUDA 内核实现，使用 CUTLASS 块缩放方法。
- python/sglang/jit_kernel/csrc/moe/expert_specialization/es_sm100_mxfp8_blockscaled_moe_group_gemm.cuh（模块 CUDA 内核；类别 other；类型 dependency-wiring）：MoE 组 GEMM 内核，包含预计算核和 grouped gemm 启动。

关键符号：es_sm100_mxfp8_blockscaled_grouped_quant, es_sm100_mxfp8_blockscaled_moe_grouped_gemm, _jit_es_sm100_mxfp8_blockscaled_group_quant, _jit_es_sm100_mxfp8_blockscaled_moe_group_gemm, _prepare_case, benchmark, test_es_sm100_mxfp8_blockscaled_grouped_mm

关键源码片段

python/sglang/jit_kernel/mxfp8.py

JIT 封装入口，提供量化和 GEMM 的高层接口，串联所有 CUDA 内核。

```
# JIT 编译函数（每个 dtype 只编译一次）
@cache_once
def _jit_es_sm100_mxfp8_blockscaled_group_quant(dtype: torch.dtype) -> Module:
    args = make_cpp_args(dtype)
    with _mxfp8_arch_env():
        return load_jit(
            'es_sm100_mxfp8_blockscaled_group_quant',
            *args,
            cuda_files=['moe/expert_specialization/es_sm100_mxfp8_blockscaled_group_quant.cuh']
```

```
        ,
        cuda_wrappers=[('es_sm100_mxfp8_blockscaled_group_quant',
                        f'EsSm100MXFP8BlockscaledGroupQuant<{args}>::run')],
        extra_dependencies=['cutlass'],
        extra_cuda_cflags=_mxfp8_cuda_flags(),
    )

@cache_once
def _jit_es_sm100_mxfp8_blockscaled_moe_group_gemm(dtype: torch.dtype) -> Module:
    args = make_cpp_args(dtype)
    with _mxfp8_arch_env():
        return load_jit(
            'es_sm100_mxfp8_blockscaled_moe_group_gemm',
            *args,
            cuda_files=['moe/expert_specialization/es_sm100_mxfp8_blockscaled_moe_group_gemm.
            cuh'],
            cuda_wrappers=[('es_sm100_mxfp8_blockscaled_moe_group_gemm',
                            f'EsSm100MXFP8BlockscaledMoeGroupGemm<{args}>::run')],
            extra_dependencies=['cutlass'],
            extra_cuda_cflags=_mxfp8_cuda_flags(),
        )

def es_sm100_mxfp8_blockscaled_grouped_quant(
    input: torch.Tensor,
    tokens_per_expert: torch.Tensor,
    expert_offsets: torch.Tensor,
    blockscale_offsets: torch.Tensor,
    quant_output: torch.Tensor,
    scale_factor: torch.Tensor,
) -> None:
    # 对拼接的 MoE 输入执行块缩放 MXFP8 量化
    module = _jit_es_sm100_mxfp8_blockscaled_group_quant(input.dtype)
    module.es_sm100_mxfp8_blockscaled_group_quant(
        input, tokens_per_expert, expert_offsets,
        blockscale_offsets, quant_output, scale_factor,
    )

def es_sm100_mxfp8_blockscaled_moe_grouped_gemm(
    a: torch.Tensor,
    b: torch.Tensor,
    sfa: torch.Tensor,
    sfb: torch.Tensor,
    expert_offsets: torch.Tensor,
    blockscale_offsets: torch.Tensor,
    tokens_per_expert: torch.Tensor,
    workspace: torch.Tensor,
    dtype: torch.dtype,
) -> torch.Tensor:
    # MoE 组 GEMM：针对每个专家独立执行矩阵乘法，使用 MXFP8 输入和输出
```

```
    num_experts, m, tokens = a.shape[0], a.shape[1], b.shape[0]
    d = torch.empty((tokens, m), device=a.device, dtype=dtype)
    # 设备端指针数组（每次调用分配，review 指出可优化）
    d_ptrs = torch.empty((num_experts,), device=a.device, dtype=torch.int64)
    b_ptrs = torch.empty((num_experts,), device=a.device, dtype=torch.int64)
    sfb_ptrs = torch.empty((num_experts,), device=a.device, dtype=torch.int64)
    module = _jit_es_sm100_mxfp8_blockscaled_moe_group_gemm(dtype)
    module.es_sm100_mxfp8_blockscaled_moe_group_gemm(
        a, b, sfa, sfb, expert_offsets, blockscale_offsets,
        tokens_per_expert, b_ptrs, sfb_ptrs, d, d_ptrs, workspace,
    )
    return d
```

评论区精华

- 类型提示 torch.Dtype 错误：gemini-code-assist[bot] 指出应改为 torch.dtype，已修复（commit 5c6f5ef）。
- 函数名 blocksclaed 拼写错误：bot 指出多处错误，已修正。
- benchmark 中 1GB workspace 重复分配：bot 建议预分配，作者回复 'Not right.' 拒绝。
- 热路径指针数组重复分配：bot 指出 es_sm100_mxfp8_blockscaled_moe_grouped_gemm 每次调用分配 d_ptrs、b_ptrs、sfb_ptrs，作者未修复，存在优化空间。
- 类型提示 torch.Dtype 错误 (correctness): 已修复（commit 5c6f5ef）
- 函数名 blocksclaed 拼写错误 (style): 已修复
- benchmark 中 1GB workspace 重复分配 (performance): 作者回复 'Not right.' 拒绝提议
- 热路径指针数组重复分配 (performance): 作者未直接回复，该问题未修复

风险与影响

- 风险：
  - 硬件兼容性：仅支持 SM100+（Blackwell）且需要 CUDA 12.8+，旧硬件将抛出异常。
  - 性能风险：JIT 编译首次调用有额外开销，但通过 @cache_once 只发生一次。
  - 正确性：测试覆盖 8/16/32/64 expert 和随机 m（1-512），diff < 0.001，但未覆盖所有边界（如 m > 512）。
  - 维护复杂性：CUDA 内核依赖 CUTLASS 模板，代码易于出错且调试困难。
  - 热路径指针分配优化未解决：每次 GEMM 调用分配设备指针数组，可能影响频繁调用场景。
- 影响：
  - 用户：Blackwell GPU 且使用 MXFP8 量化的用户可获得 5-15% 延迟降低；其他硬件用户不受影响。
  - 系统：扩展 jit_kernel 模块，新增约 1400 行代码，需维护 CUTLASS 依赖。
  - 团队：新增 JIT 内核开发和维护成本，但提供更灵活的优化路径；后续 PR 将集成此内核到 MoE 推理流程。

- 风险标记：Blackwell 独占，热路径分配问题

关联脉络

- 暂无明显关联 PR