PR #38608 完整报告

vllm-project/vllm

[XPU] Enable sequence parallel support for XPU

合并时间：2026-06-15 10:26

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/38608

执行摘要

- 一句话：为 XPU 平台启用序列并行支持
- 推荐动作：该 PR 是 XPU 平台性能优化的重要一步，改动清晰且测试完备。建议阅读其设计决策，特别是平台无关的阈值计算抽象和自定义 op 集体通信的选用考量。对于其他平台开发者，可参考其 pass_manager 导入守卫调整模式。

功能与动机

XPU 设备需要序列并行编译优化以提升推理性能。PR body 中的基准测试表明，对 Llama-2-13B 模型，开启 SP 后 TTFT 降低约 1.35%，对 Qwen3-32B 模型 TTFT 降低约 5.05%，输出 token 吞吐量也有提升。这些改善与 CUDA 平台类似，因此将 SP 支持扩展到 XPU 是合理的。

实现拆解

1. 核心阈值函数改造：在 vllm/compilation/passes/fusion/sequence_parallelism.py 中重构 get_sequence_parallelism_threshold，将原来仅检查 is_cuda() 的逻辑改为优先检查 is_xpu()，设定 XPU 特定的 min_hidden_size=4096 和 min_per_gpu_size_mb=8.0；CUDA 分支保持不变（按 capability 查询字典）；其他平台返回 None。
2. 编译管道导入调整：在 vllm/compilation/passes/pass_manager.py 中将 SequenceParallelismPass 的导入条件从 is_cuda_alike() 改为 is_cuda_alike() or is_xpu()，同时保持其他 fusion pass（如 ActivationQuantFusionPass、RMSNormQuantFusionPass）仍在 CUDA 守卫下，避免在 XPU 上引发导入错误。
3. XPU 配置清理：在 vllm/platforms/xpu.py 的 check_and_update_config 方法中，从 fusion_passes_to_disable 列表移除 "enable_sp" 条目，使序列并行在 XPU 上默认不再被禁用。
4. 测试基础设施补充：
   - 在 tests/compile/conftest.py 中新增 mock_xpu_platform fixture，模拟 XPU 平台；同时修复 mock_cuda_platform 确保在 is_cuda=False 时不会错误触发 XPU 分支。
   - 在 tests/compile/test_sequence_parallelism_threshold.py 中新增 TestGetSequenceParallelismThresholdXPU 测试类，覆盖隐藏层大小低于阈值、等于阈值、不同参数组合等场景，验证阈值计算正确性。

关键文件：

- vllm/compilation/passes/fusion/sequence_parallelism.py（模块 序列并行；类别 source；类型 core-logic；符号 get_sequence_parallelism_threshold）：核心逻辑：重构 get_sequence_parallelism_threshold 函数，添加 XPU 分支，设定阈值常量
- tests/compile/test_sequence_parallelism_threshold.py（模块 测试；类别 test；类型 test-coverage；符号 TestGetSequenceParallelismThresholdXPU, test_xpu_small_hidden_size_returns_none, test_xpu_large_model_returns_threshold, test_xpu_threshold_calculation_variations）：新增 XPU 专用测试类，覆盖阈值计算的边界和参数组合
- tests/compile/conftest.py（模块 测试辅助；类别 test；类型 test-coverage；符号 mock_xpu_platform, _mock_platform）：添加 mock_xpu_platform fixture，并修复 mock_cuda_platform 以避免 cross-contamination
- vllm/compilation/passes/pass_manager.py（模块 编译管道；类别 source；类型 dependency-wiring）：调整导入守卫，使 SequenceParallelismPass 对 XPU 可用
- vllm/platforms/xpu.py（模块 XPU 平台；类别 source；类型 core-logic）：移除禁用序列并行的配置项，使 SP 在 XPU 上默认启用

关键符号：get_sequence_parallelism_threshold, mock_xpu_platform

关键源码片段

vllm/compilation/passes/fusion/sequence_parallelism.py

核心逻辑：重构 get_sequence_parallelism_threshold 函数，添加 XPU 分支，设定阈值常量

```
def get_sequence_parallelism_threshold(
    hidden_size: int,
    tp_size: int,
    element_size: int,
) -> int | None:
    """
    计算应用序列并行的最小 token 阈值。
    返回 None 表示不应应用序列并行。
    """
    from vllm.platforms import current_platform

    # XPU 分支：使用固定的阈值常量
    if current_platform.is_xpu():
        min_hidden_size = 4096 # XPU 最小隐藏层大小
        min_per_gpu_size_mb = 8.0 # XPU 每 GPU 最小大小（MB）
    elif current_platform.is_cuda():
        capability = current_platform.get_device_capability()
        if capability is None:
            return None
        # Blackwell 系列（sm100 等）统一为 100
        if current_platform.is_device_capability_family(100):
            device_capability = 100
        else:
            device_capability = capability.to_int()
```

```
        # 从字典中查询对应 capability 的阈值
        _hidden = SP_MIN_HIDDEN_SIZE.get(device_capability)
        _gpu_mb = SP_MIN_PER_GPU_SIZE_MB.get(device_capability)
        if _hidden is None or _gpu_mb is None:
            return None
        min_hidden_size, min_per_gpu_size_mb = _hidden, _gpu_mb
    else:
        return None

    # 隐藏层太小则不启用
    if hidden_size < min_hidden_size:
        return None

    MiB = 1024 * 1024
    min_size = min_per_gpu_size_mb * MiB * tp_size
    return int(min_size // (hidden_size * element_size))
```

tests/compile/test_sequence_parallelism_threshold.py

新增 XPU 专用测试类，覆盖阈值计算的边界和参数组合

```
# XPU 专用阈值常量（必须与 sequence_parallelism.py 中的值一致）
_XPU_MIN_HIDDEN_SIZE = 4096
_XPU_MIN_PER_GPU_SIZE_MB = 8.0

class TestGetSequenceParallelismThresholdXPU:
    """测试 XPU 平台上的 get_sequence_parallelism_threshold 函数"""

    def test_xpu_small_hidden_size_returns_none(self, mock_xpu_platform):
        """隐藏层小于最小值时应返回 None"""
        with mock_xpu_platform():
            result = get_sequence_parallelism_threshold(
                hidden_size=_XPU_MIN_HIDDEN_SIZE - 1, # 比最小值小 1
                tp_size=2,
                element_size=2,
            )
        assert result is None

    def test_xpu_large_model_returns_threshold(self, mock_xpu_platform):
        """隐藏层达到最小值时应返回正确的阈值"""
        with mock_xpu_platform():
            hidden_size = _XPU_MIN_HIDDEN_SIZE
            tp_size = 2
            element_size = 2
            result = get_sequence_parallelism_threshold(
                hidden_size=hidden_size,
                tp_size=tp_size,
                element_size=element_size,
            )
        # 公式：(8 * 2 * 1024 * 1024) // (4096 * 2) = 2048
```

```
    MiB = 1024 * 1024
    expected = int(
        (_XPU_MIN_PER_GPU_SIZE_MB * tp_size * MiB) // (hidden_size * element_size)
    )
    assert result == expected
    assert result == 2048

@pytest.mark.parametrize(
    "hidden_size,tp_size,element_size,expected",
    [
        (4096, 1, 2, 1024),
        (4096, 4, 2, 4096),
        (8192, 2, 2, 1024),
        (4096, 2, 4, 1024),
    ],
)
def test_xpu_threshold_calculation_variations(
    self, mock_xpu_platform, hidden_size, tp_size, element_size, expected
):
    """验证不同参数组合下的阈值计算均正确"""
    with mock_xpu_platform():
        result = get_sequence_parallelism_threshold(
            hidden_size=hidden_size,
            tp_size=tp_size,
            element_size=element_size,
        )
    assert result == expected

def test_xpu_hidden_size_boundary(self, mock_xpu_platform):
    """在边界处（刚好小于 vs 刚好等于）行为正确"""
    with mock_xpu_platform():
        # 刚好小于最小值：返回 None
        result = get_sequence_parallelism_threshold(
            hidden_size=_XPU_MIN_HIDDEN_SIZE - 1,
            tp_size=2,
            element_size=2,
        )
        assert result is None

        # 正好等于最小值：返回非 None 阈值
        result = get_sequence_parallelism_threshold(
            hidden_size=_XPU_MIN_HIDDEN_SIZE,
            tp_size=2,
            element_size=2,
        )
        assert result is not None
```

评论区精华

1. pass_manager 导入风险：gemini-code-assist 指出将 SequenceParallelismPass 移出 is_cuda_alike() 守卫可能导致 XPU 上运行时 NameError，因为 RMSNormQuantFusionPass 仍在 CUDA 守卫中但被测试启用。jikunshang 认为不应这样改动，建议分层次导入。最终通过将 SP pass 导入条件改为 is_cuda_alike() or is_xpu()，并保留其他 pass 的 CUDA 守卫来解决。
2. 8MB 阈值参考值：yma11 询问 XPU 阈值 8MB 的参考依据。chaojun-zhang 回应暂时与 CUDA 设置一致。
3. all_reduce 自定义 op 安全性：jikunshang 质疑将 tensor_model_parallel_all_reduce 替换为 torch.ops.vllm.all_reduce.default 是否安全，是否影响其他平台或 CUDA Graph。chaojun-zhang 解释在 torch.compile 模式下必须使用自定义 op，否则 Meta tensor 会报错；同时已回退该改动，转而让 XPU 启用 use_custom_op_collectives。
4. 测试平台标记命名：ProExpertProg 建议定义统一的 skipif 标记（如 CUDA_XPU_ONLY），jikunshang 担心标记会泛滥，提议等待 RFC。最终保留内联 skipif，后续在 RFC（#39158）中统一。

- pass_manager 导入可能导致 XPU 上 NameError (correctness): 采用分层导入方案：将 SP pass 导入条件改为 is_cuda_alike() or is_xpu()，其他 pass 保持原有守卫。
- XPU 阈值 8MB 的参考依据 (question): 接受与 CUDA 一致作为临时设定，后续可根据 XPU 特性调整。
- 将 all_reduce 替换为 torch.ops.vllm.all_reduce 的安全性 (design): 回退直接替换，改为启用 use_custom_op_collectives 以使用自定义 allreduce。
- 测试平台 skipif 标记命名规范 (style): 暂不引入统一标记，维持内联 skipif，等待 RFC 结果。

风险与影响

- 风险：
  1. 编译管道依赖风险：XPU 启用了 SequenceParallelismPass，但其引用的底层通信操作（如 allreduce）依赖于 use_custom_op_collectives。如果 XPU 上的自定义 op 未正确注册或实现有差异，可能导致编译失败或运行时异常。
  2. 阈值设定未经验证：XPU 的阈值直接沿用 CUDA 的 8MB / 4096 hidden size，未经过 XPU 实际硬件特性验证，可能在特定模型上效果不佳或引入性能退化。
  3. 测试覆盖不足：添加的单元测试仅验证了阈值计算逻辑，未覆盖完整的 SP 编译和执行路径（如与 attention、norm 等 pass 的交互），无法暴露集成问题。 - 影响：影响范围：主要影响使用 Intel XPU 设备的用户，开启 SP 后首 token 时延（TTFT）预计降低 1-5%，吞吐量略有提升。对其他平台（CUDA、ROCm、CPU）无影响，因为路径不变。易用性：对用户透明，无需额外配置（SP 默认启用，由阈值自动判断）。维护成本：保持一套阈值逻辑，XPU 和 CUDA 共享计算流程，但需同步更新阈值配置。
- 风险标记：XPU 平台依赖，阈值沿用 CUDA 未验证，集成测试缺乏

关联脉络

- 暂无明显关联 PR