PR #27313 完整报告

sgl-project/sglang

[2/n] [CP] Add context parallel strategy abstractions

合并时间：2026-06-16 15:20

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27313

执行摘要

- 一句话：新增上下文并行策略抽象层
- 推荐动作：值得精读，尤其是 ContextParallelStrategy 抽象基类和包组织结构。该设计为后续模型无关的 CP 分片打下基础，是 CP 重构的核心抽象。关注后续的 #27314 等 PR 以了解逐步填充的实现。

功能与动机

现有 Context Parallelism (CP) 的实现分散在模型文件和注意力后端中，server args 混乱、模型覆盖率不对称、通信代码散落。为系统化重构，需要引入 strategy-owned runtime abstraction，将 CP 分片逻辑统一到策略接口中。详见 Roadmap Issue #27252。

实现拆解

1. 创建 CP 包和核心抽象：在 python/sglang/srt/layers/cp/ 下创建包，base.py 定义 ContextParallelStrategyKind 枚举（NONE/ZIGZAG/INTERLEAVE）、CPAttentionBackendKind 枚举、BaseContextParallelMetadata 数据类，以及 ContextParallelStrategy 抽象基类，声明 can_apply、build_metadata、shard_hidden_states、shard_position_ids、gather_hidden_states、gather_kv_cache 等抽象方法。
2. 实现策略 Shell：zigzag.py 实现 ZigzagCPStrategy，interleave.py 实现 InterleaveCPStrategy。当前仅实现 can_apply（检查 cp_size 和 token 数）和 build_metadata（返回基本元数据），其余方法 raise NotImplementedError，留待后续 PR。
3. 环境变量门控：在 environ.py 中添加 SGLANG_ENABLE_CP_V2（默认 False），并保留别名 SGLANG_ENABLE_REFACTOR_CP 作为兼容。新增 use_cp_v2() 函数（后因 review 删除，改用 is_cp_enabled 替代）。
4. 集成 server args：在 server_args.py 的 _handle_context_parallelism 方法末尾调用 init_cp_strategy(self)，从规范化后的 server args 初始化策略单例。同时添加 cp_strategy 字段映射和验证。
5. 公共导出与门面：在 __init__.py 和 utils.py 中集中导出所有公共符号，提供 is_cp_enabled、is_zigzag、is_interleave、get_cp_strategy、get_cp_strategy_kind 等便捷函数。

6. 测试覆盖：新增 test/registered/cp/test_cp_strategy_unit.py，测试枚举解析、策略初始化、门控 SGLANG_ENABLE_CP_V2 兼容性；扩展 test_server_args.py 验证 context parallel handler 会初始化策略。

关键文件：

- python/sglang/srt/layers/cp/base.py（模块 CP 抽象层；类别 source；类型 dependency-wiring；符号 ContextParallelStrategyKind, from_string, cli_value, CPAttentionBackendKind）：核心文件，定义策略枚举、注意力后端枚举、基础元数据类型和 ContextParallelStrategy 抽象基类，包含所有抽象方法签名和进程级单例辅助函数。
- python/sglang/srt/layers/cp/zigzag.py（模块 CP 抽象层；类别 source；类型 dependency-wiring；符号 ZigzagContextParallelMetadata, ZigzagCPStrategy, can_apply, build_metadata）：Zigzag 策略的具体实现壳，包括 ZigzagContextParallelMetadata 数据类和 ZigzagCPStrategy 类，展示策略接口的初步填充。
- python/sglang/srt/layers/cp/interleave.py（模块 CP 抽象层；类别 source；类型 dependency-wiring；符号 InterleaveContextParallelMetadata, InterleaveCPStrategy, can_apply, build_metadata）：Interleave 策略的具体实现壳，结构类似 zigzag.py，展示另一策略的骨架。
- python/sglang/srt/layers/cp/utils.py（模块 CP 抽象层；类别 source；类型 dependency-wiring）：公共导入门面，导出所有关键类型和辅助函数，便于外部模块引用。
- python/sglang/srt/layers/cp/__init__.py（模块 CP 抽象层；类别 source；类型 dependency-wiring）：包初始化文件，确保从 sglang.srt.layers.cp 直接导入主要符号。
- python/sglang/srt/environ.py（模块 环境配置；类别 source；类型 core-logic）：添加 SGLANG_ENABLE_CP_V2 环境变量门控，控制是否启用新的 CP v2 策略路由。
- python/sglang/srt/server_args.py（模块 服务配置；类别 source；类型 dependency-wiring）：在 _handle_context_parallelism 中调用 init_cp_strategy，确保服务器启动时根据规范化的参数初始化策略单例。
- test/registered/cp/test_cp_strategy_unit.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestCPStrategyUnit, tearDown, test_strategy_kind_maps_cli_values, test_init_cp_strategy_binds_zigzag_strategy）：单元测试文件，验证枚举解析、策略初始化、门控环境变量兼容性，保证基础功能正确。
- test/registered/unit/server_args/test_server_args.py（模块 服务配置；类别 test；类型 test-coverage；符号 test_context_parallel_handler_initializes_cp_strategy）：扩展现有 server args 测试，验证 context parallel handler 会调用 init_cp_strategy。

关键符号：ContextParallelStrategyKind.from_string, ContextParallelStrategyKind.cli_value, CPAttentionBackendKind.from_string, ContextParallelStrategy.init, ContextParallelStrategy.cp_rank, ContextParallelStrategy.can_apply, ContextParallelStrategy.build_metadata, ContextParallelStrategy.shard_hidden_states, ContextParallelStrategy.shard_position_ids, ContextParallelStrategy.gather_hidden_states, ContextParallelStrategy.gather_kv_cache, ZigzagCPStrategy.can_apply, ZigzagCPStrategy.build_metadata, InterleaveCPStrategy.can_apply,

InterleaveCPStrategy.build_metadata, init_cp_strategy, get_cp_strategy, get_cp_strategy_kind, is_cp_enabled, is_zigzag, is_interleave, TestCPStrategyUnit.test_strategy_kind_maps_cli_values, TestCPStrategyUnit.test_init_cp_strategy_binds_zigzag_strategy, TestCPStrategyUnit.test_get_cp_strategy_is_initialized_under_cp_v1_and_cp_v2

关键源码片段

python/sglang/srt/layers/cp/zigzag.py

Zigzag 策略的具体实现壳，包括 ZigzagContextParallelMetadata 数据类和 ZigzagCPStrategy 类，展示策略接口的初步填充。

```
# (ZigzagCPStrategy 部分实现 )
class ZigzagCPStrategy(ContextParallelStrategy):
    name = "zigzag"
    kind = ContextParallelStrategyKind.ZIGZAG

    def can_apply(self, num_tokens: int, forward_batch) -> bool:
        # 要求 cp_size > 1 且 token 数至少为 2 * cp_size
        if self.cp_size <= 1 or num_tokens < self.cp_size * 2:
            return False
        forward_mode = getattr(forward_batch, "forward_mode", None)
        # 仅允许在 context parallel extend 模式下应用
        return forward_mode is None or forward_mode.is_context_parallel_extend()

    def build_metadata(
        self,
        num_tokens: int,
        seqs_len: Optional[List[int]],
        extend_seqs_len: Optional[List[int]] = None,
    ) -> ZigzagContextParallelMetadata:
        # 构建基础的元数据，后续 PR 会填充详细的分拆信息
        return ZigzagContextParallelMetadata(
            total_seq_lens=sum(extend_seqs_len or seqs_len or [num_tokens]),
            bs=len(extend_seqs_len or seqs_len or [num_tokens]),
        )

    # 以下方法留待后续 PR 实现
    def shard_hidden_states(self, x: Any, forward_batch) -> Any:
        raise NotImplementedError("Zigzag hidden-state sharding will land in a follow-up PR")

    def shard_position_ids(self, positions: Any, forward_batch) -> Any:
        raise NotImplementedError("Zigzag position-id sharding will land in a follow-up PR")

    def gather_hidden_states(self, x: Any, forward_batch, stream: Optional[Any] = None) -> Any:
        raise NotImplementedError("Zigzag hidden-state gather will land in a follow-up PR")

    def gather_kv_cache(self, x: Any, forward_batch, stream: Optional[Any] = None) -> Any:
```

```
        raise NotImplementedError("Zigzag KV gather will land in a follow-up PR")

    def run_attention(self, q: Any, forward_batch, device: Any, attn_fn,
                      attention_backend: CPAttentionBackendKind = CPAttentionBackendKind.FLASH_
                      ATTENTION) -> Any:
        raise NotImplementedError("Zigzag attention dispatch will land in a follow-up PR")
```

python/sglang/srt/layers/cp/interleave.py

Interleave 策略的具体实现壳，结构类似 zigzag.py，展示另一策略的骨架。

```
# (InterleaveCPStrategy 部分实现 )
class InterleaveCPStrategy(ContextParallelStrategy):
    name = "interleave"
    kind = ContextParallelStrategyKind.INTERLEAVE

    def can_apply(self, num_tokens: int, forward_batch) -> bool:
        # 要求 cp_size > 1 且 token 数至少为 cp_size
        if self.cp_size <= 1 or num_tokens < self.cp_size:
            return False
        forward_mode = getattr(forward_batch, "forward_mode", None)
        return forward_mode is None or forward_mode.is_context_parallel_extend()

    def build_metadata(
        self,
        num_tokens: int,
        seqs_len: Optional[List[int]],
        extend_seqs_len: Optional[List[int]] = None,
    ) -> InterleaveContextParallelMetadata:
        return InterleaveContextParallelMetadata(
            total_seq_lens=sum(extend_seqs_len or seqs_len or [num_tokens]),
            bs=len(extend_seqs_len or seqs_len or [num_tokens]),
        )

    # 以下方法留待后续 PR 实现
    def shard_hidden_states(self, x: Any, forward_batch) -> Any:
        raise NotImplementedError("Interleave hidden-state sharding will land in a follow-up PR")

    def shard_position_ids(self, positions: Any, forward_batch) -> Any:
        raise NotImplementedError("Interleave position-id sharding will land in a follow-up PR")

    def gather_hidden_states(self, x: Any, forward_batch, stream: Optional[Any] = None) -> Any:
        raise NotImplementedError("Interleave hidden-state gather will land in a follow-up PR")

    def gather_kv_cache(self, x: Any, forward_batch, stream: Optional[Any] = None) -> Any:
        raise NotImplementedError("Interleave KV gather will land in a follow-up PR")
```

test/registered/cp/test_cp_strategy_unit.py

单元测试文件，验证枚举解析、策略初始化、门控环境变量兼容性，保证基础功能正确。

```
import unittest
from types import SimpleNamespace
from unittest.mock import patch

from sglang.srt.layers.cp.base import (
    ContextParallelStrategyKind,
    get_cp_strategy,
    get_cp_strategy_kind,
    init_cp_strategy,
    is_cp_enabled,
    is_interleave,
    is_zigzag,
)
from sglang.test.ci.ci_register import register_cpu_ci
from sglang.test.test_utils import CustomTestCase

register_cpu_ci(est_time=2, suite="base-a-test-cpu")


class TestCPStrategyUnit(CustomTestCase):
    def tearDown(self):
        # 每次测试后关闭 CP，避免影响其他测试
        init_cp_strategy(SimpleNamespace(enable_prefill_cp=False))

    def test_strategy_kind_maps_cli_values(self):
        # 验证枚举值与 CLI 字符串的映射
        self.assertEqual(ContextParallelStrategyKind.NONE.value, 0)
        self.assertEqual(
            ContextParallelStrategyKind.from_string("zigzag"),
            ContextParallelStrategyKind.ZIGZAG,
        )
        self.assertEqual(
            ContextParallelStrategyKind.from_string("interleave"),
            ContextParallelStrategyKind.INTERLEAVE,
        )
        self.assertEqual(ContextParallelStrategyKind.ZIGZAG.cli_value, "zigzag")
        self.assertEqual(ContextParallelStrategyKind.INTERLEAVE.cli_value, "interleave")

    def test_init_cp_strategy_binds_zigzag_strategy(self):
        # 模拟启用 CP 并选用 zigzag 策略，验证单例状态
        init_cp_strategy(
            SimpleNamespace(
                enable_prefill_cp=True,
                cp_strategy="zigzag",
                attn_cp_size=4,
            )
        )
        self.assertTrue(is_cp_enabled())
        self.assertTrue(is_zigzag())
```

```
        self.assertFalse(is_interleave())
        self.assertEqual(get_cp_strategy_kind(), ContextParallelStrategyKind.ZIGZAG)

    def test_get_cp_strategy_is_initialized_under_cp_v1_and_cp_v2(self):
        # 验证在 CP v1（SGLANG_ENABLE_CP_V2=False）和 CP v2 下策略均被初始化
        init_cp_strategy(
            SimpleNamespace(
                enable_prefill_cp=True,
                cp_strategy="interleave",
                attn_cp_size=4,
            )
        )
        with patch(
            "sglang.srt.environ.envs.SGLANG_ENABLE_CP_V2.get", return_value=False
        ):
            self.assertIsNotNone(get_cp_strategy())
            self.assertTrue(is_cp_enabled())
            self.assertTrue(is_interleave())

        with patch(
            "sglang.srt.environ.envs.SGLANG_ENABLE_CP_V2.get", return_value=True
        ):
            self.assertIsNotNone(get_cp_strategy())


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

评论区精华

文件命名与组织：作者将 strategy.py 重命名为 utils.py，并将初始化逻辑、use_cp_v2 等函数移入 base.py，简化文件结构。同时明确了 shard_hidden_states/gather_hidden_states 等方法的命名和职责（在 base.py 中关联评论）。

CP v2 门控简化：最初 use_cp_v2() 函数被提出，但经 review 认为可直接通过 is_cp_enabled() 覆盖，故移除。SGLANG_ENABLE_CP_V2 改为直接使用 EnvBool 而非 EnvBoolWithAlias。

添加策略说明图：要求在 zigzag.py 和 interleave.py 文件头部添加 ASCII 图表说明分片方式，从旧 cp_utils.py 迁移。

测试适应性：测试中确保 get_cp_strategy() 在 CP v1 和 CP v2 下均返回非 None，且 init_cp_strategy 在两种情况下均被调用。

- 文件命名与组织 (design): 已执行：文件重命名、函数移动、copyright 更新。
- CP v2 门控简化 (design): 已移除 use_cp_v2，改用 is_cp_enabled；环境变量改为 EnvBool。
- 添加策略说明 ASCII 图 (documentation): 已添加图表说明，用户可直观了解分片方案。
- 抽象方法命名与职责明确 (design): 已按建议重命名并更新 docstring。

- 测试覆盖门控分支 (testing): 测试已补充 test_get_cp_strategy_is_initialized_under_cp_v1_and_cp_v2，使用 patch 模拟环境变量。

风险与影响

- 风险：兼容性风险：新抽象默认关闭，现有 CP 路径完全不受影响，风险低。 未实现的方法：shard_hidden_states 等方法抛出 NotImplementedError，若在 CP v2 路由启用前被意外调用将崩溃，但通过后续 PR 逐步填充可避免。 环境变量别名：SGLANG_ENABLE_REFACTOR_CP 作为已废弃别名保留，但删除旧环境变量的用户无感知。
- 影响：用户：无用户可见变更，现有命令行参数继续生效。后续 CP v2 启用后将影响 CP 行为。 系统：增加少量内存开销（策略单例），无性能影响。 团队：为后续 CP 重构提供统一接口，降低模型接入成本。
- 风险标记：新抽象默认关闭，未实现方法可能误调，环境变量别名兼容，测试覆盖率有限，CP v2 尚未启用

关联脉络

- PR #27312 [1/n] [CP] Simplify prefill context parallel server args: 本 PR #27313 依赖于 #27312，后者简化了 server args 并规范化了 cp_strategy 字段。本 PR 在此基础上初始化策略单例。
- PR #27252 [Roadmap] Prefill Context Parallel Refactor: 本 PR 是 Roadmap issue #27252 的第二项，整体设计遵循该 issue。
- PR #27314 [3/n] [CP] Integrate CP-v2 strategy into model runner: 预期的后续 PR（未在历史中明确，但根据 roadmap 推测），将策略抽象集成到模型 runner。