PR #36887 完整报告

sgl-project/sglang

[CI] Slim JIT kernel unit tests

合并时间：2026-08-29 07:26

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36887

执行摘要

- 一句话：引入 fork 预加载工作进程并精简 JIT 内核单测矩阵
- 推荐动作：值得精读，尤其是 fork worker 的进程 / 管道协议设计、退出码规范化和 get_ci_test_range 的代表性用例选取方法；该模式可复制到其他长时间运行的 kernel 单测套件，是 CI 性能治理的典型案例。

功能与动机

H100 JIT kernel unit-test 步骤开始命中 30 分钟超时；最近一次成功的未拆分运行耗时 27m59s，并约以独立 Python 进程启动 100 个测试文件。与其调高超时，不如降低套件本身耗时；这与 #36775 的分区思路互补。

实现拆解

1. 新增 python/sglang/test/ci/fork_test_worker.py：_preload_common_modules 预加载 numpy/scipy/pytest/torch/triton 且保持 CUDA-free；run_file_in_fork 在 os.fork() 子进程中用 runpy.run_path 执行测试文件，并把 SystemExit 规范化为 0..255 退出码；main 通过 stdin 的 JSON 行接收命令、通过专用 fd 回传 returncode 与 elapsed。这一层复用了预加载解释器，减少文件间重复导入开销。
2. 在 python/sglang/test/ci/ci_utils.py 中加入 _ForkTestWorker 父进程封装和 run_unittest_files 的 fork_worker_batch_size 参数；参数大于 1 时启用 fork worker，保留逐文件 fail-fast、超时、重试与 TIMINGS 上报，默认值 1 维持原有逐文件 subprocess 行为。工作进程每服务 fork_worker_batch_size 个文件后重启，避免长期运行的 Python 状态累积。
3. 压缩五个热点测试矩阵：test_hadamard_jit.py 145 -> 29（est_time 128s -> 32s）、attention/test_rope.py 230 -> 26（64s -> 24s）、diffusion/test_rope.py 144 -> 10、test_per_token_group_quant.py 51 -> 35（90s -> 65s）、test_sconv_extend_metadata.py 50 -> 14。完整笛卡尔积通过 get_ci_test_range 保留在 nightly/full 运行，nightly 通过 SGLANG_JIT_KERNEL_RUN_FULL_TESTS=1 展开。
4. CI 接入与测试配套：.github/workflows/pr-test-jit-kernel.yml 在 H100 JIT kernel unit lane 上启用 fork worker；test/run_suite.py 增加批处理参数传递；新增 test/registered/unit/test_fork_test_worker.py 的 CPU 协议测试，验证连续文件在隔离子进程中运行且 builtins marker 与环境变量不泄漏。

关键文件：

- python/sglang/test/ci/fork_test_worker.py（模块 测试工作进程；类别 test；类型 test-coverage；符号 _preload_common_modules, _normalize_exit_code, _run_file, _wait_status_to_returncode）：新增的 fork worker 是本次性能优化的核心：预加载公共依赖后在隔离子进程中执行测试文件，避免重复导入开销。
- python/sglang/test/ci/ci_utils.py（模块 CI 调度；类别 test；类型 test-coverage；符号 _ForkTestWorker, init, run, close）：集成 fork worker 并扩展 run_unittest_files 支持批量 fork 执行，默认保持原有逐文件行为，是 CI 逻辑接入的关键。
- test/registered/unit/test_fork_test_worker.py（模块 隔离测试；类别 test；类型 test-coverage；符号 TestForkTestWorker, test_files_run_in_isolated_children）：新增 CPU 协议测试，验证连续测试文件在隔离的 fork 子进程中运行，防止模块 / 环境状态泄漏。
- test/registered/kernels/ops/quantization/test_hadamard_jit.py（模块 Hadamard 内核；类别 test；类型 test-coverage；符号 _mn_cases, test_hadamard_transform, test_hadamard_transform_non_power_of_two）：Hadamard 变换的 PR 参数矩阵从 145 例减至 29 例，是本次矩阵缩减的主要收益来源之一。
- test/registered/kernels/ops/attention/test_rope.py（模块 RoPE 内核；类别 test；类型 test-coverage）：attention RoPE 的参数量从 230 组缩减到 26 组，显著降低热点文件耗时。
- test/registered/kernels/ops/quantization/test_per_token_group_quant.py（模块 分组量化；类别 test；类型 test-coverage；符号 test_masked, test_fp32_scale）：per-token group quant 的 PR 矩阵从 51 例缩至 35 例，保留 bit-exact 边界。
- test/registered/kernels/ops/mamba/test_sconv_extend_metadata.py（模块 SConv 元数据；类别 test；类型 test-coverage）：sconv extend metadata 的 PR 矩阵从 50 例缩减到 14 例。
- test/registered/kernels/ops/diffusion/test_rope.py（模块 扩散 RoPE；类别 test；类型 test-coverage）：diffusion QK-norm RoPE 参数扫描从 144 例减到 10 例。
- test/run_suite.py（模块 套件运行；类别 test；类型 test-coverage）：在本地运行 / 测试套件中接入 fork worker 批次逻辑，但默认不改变原有执行方式。
- .github/workflows/pr-test-jit-kernel.yml（模块 CI 工作流；类别 infra；类型 infrastructure）：CI 工作流启用 fork worker，接入精简后的内核单测任务。

关键符号：_preload_common_modules, _normalize_exit_code, _run_file, _wait_status_to_returncode, run_file_in_fork, main, _ForkTestWorker, run_unittest_files, test_files_run_in_isolated_children

关键源码片段

python/sglang/test/ci/ci_utils.py

集成 fork worker 并扩展 run_unittest_files 支持批量 fork 执行，默认保持原有逐文件行为，是 CI 逻辑接入的关键。

```
class _ForkTestWorker:
    # 预加载式 fork worker 的父进程侧封装：
```

```
# 通过管道向 fork_test_worker.py 发文件名命令，并逐行读回 JSON 结果。
def __init__(self):
    result_read_fd, result_write_fd = os.pipe()
    worker_path = os.path.join(os.path.dirname(__file__), "fork_test_worker.py")
    self.process = subprocess.Popen(
        ["python3", worker_path, "--result-fd", str(result_write_fd)],
        stdin=subprocess.PIPE,
        stdout=None,
        stderr=None,
        text=True,
        pass_fds=(result_write_fd,),
    )
    os.close(result_write_fd)
    self.result_stream = os.fdopen(result_read_fd)
    self.files_run = 0

def run(self, filename: str) -> tuple[int, float]:
    tic = time.perf_counter()
    if self.process.poll() is not None or self.process.stdin is None:
        return 1, 0.0
    try:
        self.process.stdin.write(json.dumps({"filename": filename}) + "\n")
        self.process.stdin.flush()
        result_line = self.result_stream.readline()
    except (BrokenPipeError, OSError):
        return 1, time.perf_counter() - tic
    if not result_line:
        return 1, time.perf_counter() - tic
    try:
        result = json.loads(result_line)
    except json.JSONDecodeError:
        return 1, time.perf_counter() - tic
    self.files_run += 1
    return int(result["returncode"]), float(result["elapsed"])

def close(self, terminate: bool = False):
    # 优先用 stop 命令优雅退出，超时 / 断管时杀进程树兜底
    if self.process.poll() is None:
        if terminate:
            kill_process_tree(self.process.pid)
        elif self.process.stdin is not None:
            try:
                self.process.stdin.write(json.dumps({"command": "stop"}) + "\n")
                self.process.stdin.flush()
                self.process.wait(timeout=10)
            except (BrokenPipeError, subprocess.TimeoutExpired):
                kill_process_tree(self.process.pid)
    if self.process.poll() is None:
        self.process.kill()
```

```
        try:
            self.process.wait(timeout=10)
        except subprocess.TimeoutExpired:
            self.process.kill()
            self.process.wait()
        if self.process.stdin is not None:
            self.process.stdin.close()
        self.result_stream.close()
```

test/registered/unit/test_fork_test_worker.py

新增 CPU 协议测试，验证连续测试文件在隔离的 fork 子进程中运行，防止模块 / 环境状态泄漏。

```
# CPU 协议测试：验证 fork worker 的子进程隔离语义
@unittest.skipUnless(hasattr(os, "fork"), "fork requires a POSIX platform")
class TestForkTestWorker(CustomTestCase):
    def test_files_run_in_isolated_children(self):
        result_read_fd, result_write_fd = os.pipe()
        process = subprocess.Popen(
            [
                sys.executable,
                fork_test_worker.__file__,
                "--result-fd",
                str(result_write_fd),
            ],
            stdin=subprocess.PIPE,
            text=True,
            pass_fds=(result_write_fd,),
        )
        os.close(result_write_fd)

        try:
            with (
                tempfile.TemporaryDirectory() as tmpdir,
                os.fdopen(result_read_fd) as result_stream,
            ):
                first = Path(tmpdir) / "first.py"
                first.write_text(
                    "import builtins\n"
                    "import os\n"
                    "builtins._sglang_fork_worker_marker = 41\n"
                    "os.environ['SGLANG_FORK_WORKER_TEST'] = 'leaked'\n"
                    "raise SystemExit(0)\n"
                )
                second = Path(tmpdir) / "second.py"
                second.write_text(
                    "import builtins\n"
                    "import os\n"
                    "assert not hasattr(builtins, '_sglang_fork_worker_marker')\n"
```

```
                "assert 'SGLANG_FORK_WORKER_TEST' not in os.environ\n"
                "raise SystemExit(3)\n"
            )

            results = []
            # 连续运行两个文件，校验各自的退出码与耗时
            for filename in (first, second):
                process.stdin.write(json.dumps({"filename": str(filename)}) + "\n")
                process.stdin.flush()
                results.append(json.loads(result_stream.readline()))

            # 第一个文件返回 0，第二个文件返回 3，说明子进程互不污染
            self.assertEqual([result["returncode"] for result in results], [0, 3])
            self.assertTrue(all(result["elapsed"] >= 0 for result in results))

            process.stdin.write(json.dumps({"command": "stop"}) + "\n")
            process.stdin.flush()
            self.assertEqual(process.wait(timeout=30), 0)
        finally:
            if process.poll() is None:
                process.kill()
                process.wait()
```

评论区精华

该 PR 没有 review 评论线程；从 6 个提交可读出两个关键决策：其一是 fix: keep fork preloader CUDA-free，说明早期版本曾触发 CUDA 初始化，后通过显式检查修正；其二是 ci: trim sconv metadata PR matrix 与 test: document full sconv nightly coverage，表明“PR 快扫 + nightly 全扫”的分层覆盖策略是刻意设计。

- 暂无高价值评论线程

风险与影响

- 风险：
  - 预加载后 fork 的兼容性风险：如果未来预加载栈意外初始化 CUDA，_preload_common_modules 会直接抛 RuntimeError，整条 CI lane 失败；要求后续维护者保持预加载部分保持 CUDA-free。
  - 执行语义差异：fork 子进程用 runpy.run_path 执行测试文件，与直接 python3 file.py -f 在 __file__、参数解析、pytest 插件加载上可能有细微差别，个别测试可能在 CI 上表现不同。
  - 矩阵覆盖缩减：PR 阶段只跑代表性组合，部分边界 bug 可能要等到 nightly 才暴露；依赖 get_ci_test_range 的选样质量。
  - 平台限制：os.fork 仅 POSIX 可用，入口已有 hasattr(os, "fork") 防护，Windows 上不会启用 fork worker。
- 影响：

- 开发者：H100 JIT kernel unit lane 预计从约 28 分钟缩短到约 10 分钟，明显减少 PR 测试等待时间。
- 系统：新增一个可选的多进程执行路径，但默认关闭，其他套件行为不变，影响面仅限显式启用 fork worker 的 lane。
- 团队：提供了一个可复用的“预加载 + fork 隔离”测试执行模式，后续可推广到其他慢速内核单测套件。
- 风险标记：CI 超时风险，fork 隔离依赖平台，矩阵覆盖缩减，管道协议异常路径

关联脉络

- PR #36775 （输入中未提供，标题未知）：PR body 明确说明两者互补：该 PR 通过分区降低墙钟时间，本 PR 降低 H100 总耗时；两者共同解决 JIT kernel 单元测试超时问题。