PR #22416 完整报告

sgl-project/sglang

[Apple Silicon] [MLX] MLX decode partial overlap scheduling for generation (async eval)

合并时间：2026-04-30 03:21

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/22416

执行摘要

- 一句话：MLX 后端实现解码异步重叠调度
- 推荐动作：值得精读。该 PR 展示了如何利用 MLX 的 lazy evaluation 特性设计高效的 GPU 流水线，是 Apple Silicon 推理性能优化的核心里程碑。SchedulerMlxOverlapMixin 中的链式调度设计（两图链、链中断条件、async_eval 与 finalize 分离）具有较高参考价值。后续可以考虑扩展到 prefill/extend 链以及更鲁棒的 KV 缓存管理。

功能与动机

MLX 后端原有的实现中，每次解码步骤都会导致 CPU 与 GPU 同步，产生 GPU 空闲间隙（如 PR body 中的截图所示），限制了吞吐。该 PR 参考了 SGLang CUDA 版本的重叠调度设计（参见 https://www.lmsys.org/blog/2024-12-04-sglang-v0-4/），旨在通过异步评估消除这些空闲间隙，提升 Apple Silicon 上的生成性能。关联 Issue #22114 和 #22466。

实现拆解

1. 新增 SchedulerMlxOverlapMixin（python/sglang/srt/hardware_backend/mlx/scheduler_mixin.py）：包含 event_loop_overlap_mlx 主循环，维护两个 in-flight 的 MLX 计算图（pending_curr 和 pending_next），利用 mx.async_eval 实现 GPU 流水线。定义了 MlxPendingJob dataclass 持有未完成的 lazy 工作。
2. 拆分 MlxModelRunner（python/sglang/srt/hardware_backend/mlx/model_runner.py）为 lazy API：新增 prefill_start/prefill_finalize、extend_start/extend_finalize、decode_batch_start/decode_batch_finalize 及 decode_batch_start_chained。新增 MlxPendingPrefill、MlxPendingExtend、MlxPendingDecode 数据类持有 lazy 结果。_cache_state_arrays 工具方法用于展平缓存数组，便于 mx.async_eval。
3. 在 MlxTpModelWorker（python/sglang/srt/hardware_backend/mlx/tp_worker.py）中增加异步方法：async_forward_batch_generation_mlx 返回 lazy 结果，async_chained_decode_mlx 在上一 decode 的 lazy 输出上构建下一步计算图，finalize_mlx_result 阻塞等待并产生 GenerationBatchResult。提取 _cleanup_stale_rids 辅助方法。
4. 调整调度器（python/sglang/srt/managers/scheduler.py）：混入 SchedulerMlxOverlapMixin，根据 enable_overlap_mlx 标志在 dispatch_event_loop 中启用新的事件循环路径。分离 enable_overlap 为 CUDA 和 MLX 两个控制变量。init_overlap 中为 MLX 跳过 CUDA 流管理。

5. 修改服务器参数（python/sglang/srt/server_args.py）：调整 _handle_mps_backends 逻辑：仅当不使用 MLX 时才禁用 overlap schedule（因为 MLX 默认启用 overlap）。
6. 修改输出处理器（python/sglang/srt/managers/scheduler_output_processor_mixin.py）：process_batch_result_decode 中支持 MLX 路径的列表类型 next_token_ids，并在 finished 请求处理时同时检查 enable_overlap_mlx。
7. 文档更新：在 apple_metal.mdx 和 apple_metal.md 中添加重叠调度功能说明和 benchmark 命令。

关键文件：

- python/sglang/srt/hardware_backend/mlx/scheduler_mixin.py（模块 调度器混入；类别 source；类型 core-logic；符号 MlxPendingJob, SchedulerMlxOverlapMixin, event_loop_overlap_mlx, _finalize）：新增的调度器混入核心文件，包含 event_loop_overlap_mlx 主循环逻辑和 MlxPendingJob 数据类。
- python/sglang/srt/hardware_backend/mlx/model_runner.py（模块 模型运行器；类别 source；类型 data-contract；符号 MlxPendingPrefill, MlxPendingExtend, MlxPendingDecode, _cache_state_arrays）：修改量最大的文件，拆分了 lazy API，定义了持有 lazy 结果的数据类。
- python/sglang/srt/hardware_backend/mlx/tp_worker.py（模块 工作进程；类别 source；类型 dependency-wiring；符号 _cleanup_stale_rids, async_forward_batch_generation_mlx, _async_extend_batch, _cache_state）：增加了异步前向和后处理函数，构成 overlap scheduler 与 model runner 之间的桥梁。

关键符号：event_loop_overlap_mlx, _finalize, _launch_fresh, _launch_chained, prefill_start, prefill_finalize, extend_start, extend_finalize, decode_batch_start, decode_batch_finalize, decode_batch_start_chained, async_forward_batch_generation_mlx, async_chained_decode_mlx, finalize_mlx_result, _cache_state_arrays, _cleanup_stale_rids

关键源码片段

python/sglang/srt/hardware_backend/mlx/scheduler_mixin.py

新增的调度器混入核心文件，包含 event_loop_overlap_mlx 主循环逻辑和 MlxPendingJob 数据类。

```
"""MLX overlap scheduling mixin for the SGLang scheduler.

Provides ``event_loop_overlap_mlx``, which pipelines MLX forward
passes by keeping two in-flight lazy graphs queued on the GPU while
the scheduler runs its CPU-side bookkeeping on the tokens of the
older one.
"""


@dataclass
class MlxPendingJob:
    """Represents an unfinished MLX forward pass queued on the GPU.
```

```
    ``lazy_tokens``: mlx.array of token IDs, not yet evaluated.
    ``prefills``/``extends``/``decode``: per-mode state for finalization.
    ``mode``: ``"decode"``, ``"extend"``, or ``"idle"``.
    ``batch_copy``: snapshot of ``ScheduleBatch`` at launch time,
    decoupled from the live batch to avoid races.
    """
    lazy_tokens: Optional[mx.array]
    prefills: list["MlxPendingPrefill"]
    extends: list["MlxPendingExtend"]
    decode: Optional["MlxPendingDecode"]
    mode: str
    batch_copy: "ScheduleBatch"
    reqs: List[Req]


class SchedulerMlxOverlapMixin:
    """Mixin that adds MLX overlap scheduling to :class:`Scheduler`."""

    @DynamicGradMode()
    def event_loop_overlap_mlx(self: "Scheduler"):
        """MLX overlap loop modelled on ``mlx_lm.generate.generate_step``.

        At steady state we keep TWO in-flight MLX graphs:
        ``pending_curr`` (about to be finalized) and
        ``pending_next`` (built on top of ``pending_curr``'s lazy output,
        already handed to ``mx.async_eval``).
        """
        # Initialize state
        pending_curr: Optional[MlxPendingJob] = None
        pending_next: Optional[MlxPendingJob] = None
        self.result_queue.clear()

        while not self.is_shutdown:
            # Finalize the previous step's pending_curr, if any
            if pending_curr is not None:
                self._finalize(pending_curr)

            # If a chained next step exists, shift it to current
            if pending_next is not None:
                pending_curr = pending_next
                pending_next = None
            else:
                # Otherwise schedule a new batch
                batch = self.get_next_batch_to_run()
                if batch is None:
                    pending_curr = None
                    continue
                pending_curr = self._launch_fresh(batch)
```

```
            # Decide whether we can chain a next decode step
            if self._can_chain(pending_curr):
                pending_next = self._try_launch_chained(pending_curr)

            # Block on pending_curr tokens to feed into bookkeeping
            # (the GPU is already running pending_next in the background)
            _ = pending_curr.lazy_tokens.tolist()
```

python/sglang/srt/hardware_backend/mlx/model_runner.py

修改量最大的文件，拆分了 lazy API，定义了持有 lazy 结果的数据类。

```
@dataclass
class MlxPendingPrefill:
    """Lazy prefill state, finalized after ``mx.eval``.
    ``cache`` is per-layer ``ContiguousKVCache`` list for commit.
    """
    lazy_token: mx.array
    cache: list # list[ContiguousKVCache]
    req_id: str
    full_token_ids: list[int]
    req_pool_idx: int
    synced_offset: int


@dataclass
class MlxPendingExtend:
    """Lazy chunked-prefill-continuation state for an existing request.
    Uses the request's existing per-layer cache.
    """
    lazy_token: mx.array
    req_id: str
    new_token_ids: list[int]
    new_synced_offset: int


@dataclass
class MlxPendingDecode:
    """Lazy decode state for a batch.
    ``caches``: per-request list of per-layer ``ContiguousKVCache``
    references that the attention wrapper writes into.
    """
    lazy_tokens: mx.array
    req_ids: list[str]
    caches: list # list[list[ContiguousKVCache]]


class MlxModelRunner:
    # ... (existing fields)

    def decode_batch_start(self, req_ids: list[str]) -> MlxPendingDecode:
```

```
        """Start a decode step without evaluating.
        Builds the compute graph, writes KV caches in-place,
        and returns lazy token output.
        """
        # ... merge KV caches, run model forward, collect lazy tokens
        # Return MlxPendingDecode without calling mx.eval
        return MlxPendingDecode(
            lazy_tokens=logits.argmax(-1),
            req_ids=req_ids,
            caches=merged_caches,
        )

    def decode_batch_start_chained(self, prev: MlxPendingDecode) -> MlxPendingDecode:
        """Launch next decode step on top of a still-lazy previous decode.
        Reuses the same cache objects, so MLX tracks the dependency.
        """
        # Build graph using prev's lazy (unevaluated) output as input
        # and the same cache lists (already updated in-place).
        return self.decode_batch_start(prev.req_ids)

    def decode_batch_finalize(self, pending: MlxPendingDecode) -> list[int]:
        """Block on lazy tokens and return token IDs.
        Evaluates tokens together with cache arrays to materialize writes.
        """
        cache_arrays = [
            arr for c_list in pending.caches for arr in self._cache_state_arrays(c_list)
        ]
        mx.eval(pending.lazy_tokens, *cache_arrays)
        return pending.lazy_tokens.tolist()
```

python/sglang/srt/hardware_backend/mlx/tp_worker.py

增加了异步前向和后处理函数，构成 overlap scheduler 与 model runner 之间的桥梁。

```
  def async_forward_batch_generation_mlx(
      self,
      model_worker_batch: ModelWorkerBatch,
  ) -> tuple[
      Union[mx.array, None],
      list[MlxPendingPrefill],
      list[MlxPendingExtend],
      Optional[MlxPendingDecode],
      str,
  ]:
      """Start an async (lazy) forward pass through the MLX model runner.

      Returns (lazy_result, prefills, extends, decode, mode) without
      blocking on the GPU. The caller can later call ``finalize_mlx_result``
      to block and produce a ``GenerationBatchResult``.
      """
```

```
    forward_mode = model_worker_batch.forward_mode
    reqs = model_worker_batch.reqs

    if forward_mode.is_idle():
        return (None, [], [], None, "idle")

    self._cleanup_stale_rids(forward_mode, {req.rid for req in reqs})

    if forward_mode.is_extend():
        # ... build lazy extend graphs, return pending state
        pass
    else:
        # Decode: use decode_batch_start
        pending_decode = self._mlx_runner.decode_batch_start(
            [req.rid for req in reqs]
        )
        return (pending_decode.lazy_tokens, [], [], pending_decode, "decode")


def async_chained_decode_mlx(self, prev_decode: MlxPendingDecode) -> MlxPendingDecode:
    """Build the next decode step on top of a still-lazy previous decode.
    Reuses the cache objects from prev_decode, so MLX tracks the
    dependency graph. The caller should hand the result to
    ``mx.async_eval`` immediately.
    """
    next_decode = self._mlx_runner.decode_batch_start_chained(prev_decode)
    # Fire async evaluation: GPU will execute this step as soon as
    # the previous step's dependencies are resolved.
    mx.async_eval(next_decode.lazy_tokens)
    return next_decode


def finalize_mlx_result(self, pending_job: MlxPendingJob) -> GenerationBatchResult:
    """Block on lazy tokens and produce a normal GenerationBatchResult.
    Depending on pending_job.mode, calls prefill/extend/decode finalize.
    """
    if pending_job.mode == "decode":
        next_token_ids = self._mlx_runner.decode_batch_finalize(pending_job.decode)
        # ... build GenerationBatchResult
    elif pending_job.mode == "extend":
        # ... merge prefills and extends
        pass
    # ... return GenerationBatchResult
```

评论区精华

1. Mixin 设计：yeahdongcn 建议将 MLX 重叠逻辑抽取为独立的 mixin 类以减少对 scheduler.py 的侵入，得到采纳。

2. enable_overlap 变量命名：yeahdongcn 提议将 CUDA 相关变量重命名为 enable_overlap_torch 或 enable_overlap_cuda，以清晰区分 MLX 和 CUDA 路径。最终保留 enable_overlap 作为通用标志，新增 enable_overlap_mlx 专门控制 MLX。
3. 导入路径错误：yeahdongcn 指出 server_args.py 中存在错误的相对导入 from python.sglang.srt...，后修正为绝对导入。
4. BatchedKVCacheManager 设计：alexnails 提出是否考虑使用持久的 KV 缓存管理器（类似 CUDA 的 BatchedKVCacheManager）以简化代码。changminbark 解释由于 MLX 的 lazy eval 和 per-request ContiguousKVCache 设计，当前实现更简单且避免了额外拷贝。
5. 链中断与请求完成：alexnails 和 Kangyan-Zhou 讨论了当请求完成时链的行为，发现 process_batch_result_decode 中的 finished-request guard 没有检查 enable_overlap_mlx，导致重复释放 KV 缓存。最终通过添加 self.enable_overlap_mlx 检查修复。
6. 性能优化前瞻：alexnails 建议将 mx.array(ctx.seq_lens) 缓存到 BatchedDecodeContext 中避免每层重复创建，changminbark 同意并放入后续 PR 中。

- Mixin 设计建议 (design): 采纳建议，创建了 SchedulerMlxOverlapMixin。
- enable_overlap 变量命名与分离 (design): 最终在 scheduler.py 中引入 enable_overlap_mlx，enable_overlap 同时用于 CUDA 和 MLX 的通用条件。
- 导入路径错误 (correctness): 修正为正确的导入路径。
- BatchedKVCacheManager 设计质疑 (design): 保持当前 per-request cache 设计，未引入 BatchedKVCacheManager。
- finished-request guard 缺少 MLX 检查 (correctness): 添加 self.enable_overlap_mlx 检查到 guard 条件中。
- seq_lens 缓存优化 (performance): changminbark 同意并将此优化放入后续 PR。

风险与影响

- 风险：
  1. 内存管理复杂性：MLX lazy evaluation 会累积未评估的计算图，需依赖 mx.clear_cache() 定期清理（_decode_step_ct 计数器触发的 mx.metal.clear_cache() 调用），若清理不当可能导致内存泄漏。
  2. 链中断导致浪费：当请求完成时，已启动的 pending_next 仍然被评估，多余的一个 token 被丢弃，造成约一步 decode 的计算浪费。
  3. 仅 decode-decode 链：预填充、扩展或 batch 组合变化会中断链，回退到标准路径，部分重叠无法覆盖。
  4. 缺少单元测试覆盖：当前没有针对 event_loop_overlap_mlx 的单元测试，仅依靠手动功能测试和 benchmark，回归风险较高。
  5. 与调度器核心架构耦合：mixinin 方式虽然减少了侵入，但仍需在 scheduler.py 的关键路径中添加条件判断，未来核心重构可能需要同步调整。 - 影响：影响范围：仅 Apple Silicon（macOS）上使用 MLX 后端的用户，其他后端（CUDA/ROCm/CPU）不受影响。影响程度：默认启用，用户可通过 --disable-overlap-schedule 或环境变量 SGLANG_USE_MLX=0 关闭。预期能显著提升连续解码吞吐（尤其是在批量较大时），

但对预填充或混合 batch 场景提升有限。文档已更新用法说明。团队：维护需要了解 MLX lazy evaluation 和 overlap 调度机制的开发者。 - 风险标记：缺少测试覆盖，内存管理复杂，链中断导致计算浪费，仅 decode-decode 链

关联脉络

- PR #21509 [Apple Silicon] RadixCache and ContiguousKVCache support: 本 PR 依赖于 #21509 引入的 RadixCache 和 ContiguousKVCache 重构，PR 作者在开发过程中多次合并该分支以适配新缓存架构。