CHINA LINUX KERNEL
中国Linux内核开发者大会

vivo

ZRAM异构压缩技术
基于GPU加速的内存回收方案

林芝驰
vivo性能优化工程师

CHINA LINUX KERNEL
中国Linux内核开发者大会

vivo

目录

CONTENTS

问题背景 - 高负载场景下的内存回收困局

➢ 在内存与CPU双高负载的场景，如影像场景，kswapd与高优先级业务线程激烈抢占CPU大核资源：

21077 items selected. CPU Slices (21077)

Name	Wall Duration	Average Wall Duration	Occurrences
kswapd0	971.813 ms	3.009 ms	323
RenderThread	269.272 ms	0.554 ms	486
exe_cq/0	234.821 ms	0.111 ms	2111
VcodecProcess	162.181 ms	1.502 ms	108
rx_thread	137.502 ms	0.224 ms	615

➢ 压缩算法（LZ4）执行时间占整个回收过程的30%~40%+，是导致CPU争抢和体验劣化的关键瓶颈：

优化探索 - 异构探索

➢ 在传统以CPU为中心化的架构中，这个问题几乎是一个‘死结’。

1. 优化CPU压缩算法？==》无法根除竞争。
2. 减少内存回收？==》应用对内存的需求只增不减，减少内存回收会让用户体验进一步劣化。

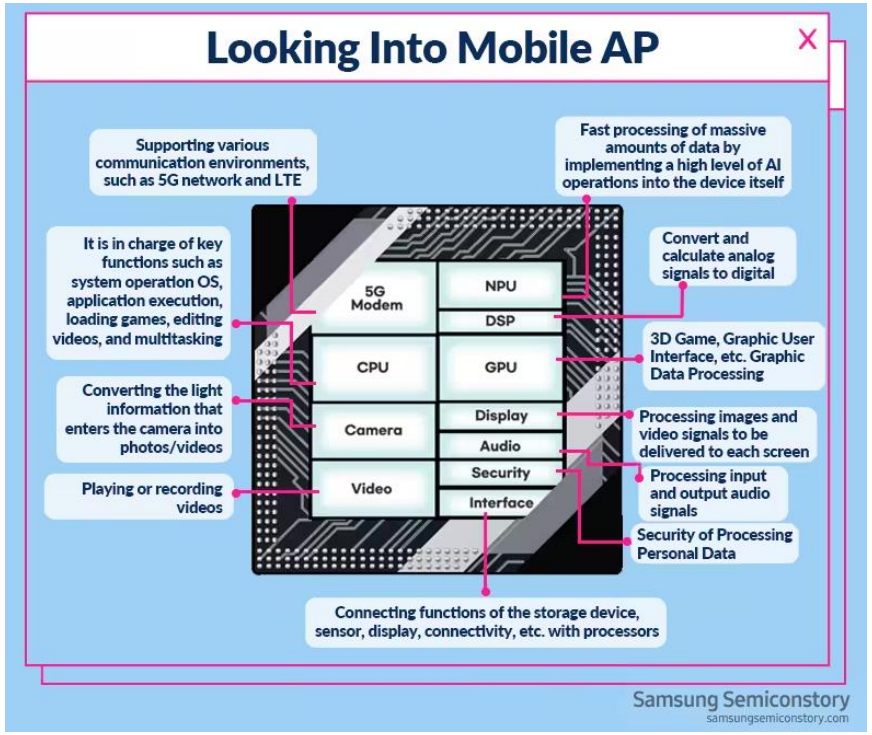
➢ SOC上能够提供算力的不仅仅是CPU，还有GPU、NPU、DSP等等。

➢ 将压缩算法迁移到其它计算单元上运行，让CPU更多地服务于关键业务线程。

社区patch：zsmalloc/zram: there be preemption

```
For instance, this makes it impossible to use async compression
algorithms or/and H/W compression algorithms, which can wait for OP
completion or resource availability.  This also restricts what
compression algorithms can do internally, for example, zstd can
allocate internal state memory for C/D dictionaries:
```

*https://lore.kernel.org/lkml/20250303022425.285971-1-senozhatsky@chromium.org/T/

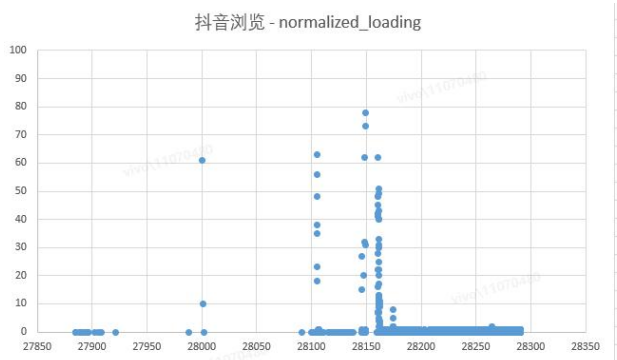


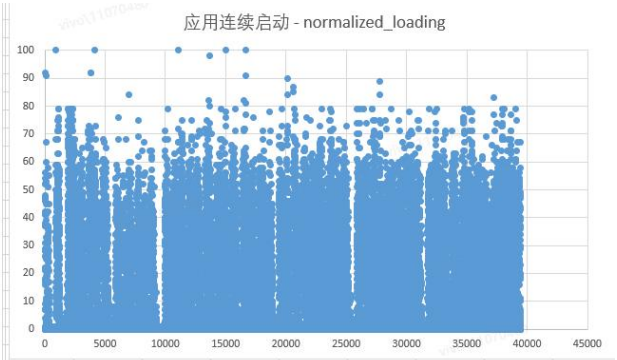
*https://semiconductor.samsung.com/support/tools-resources/dictionary/semiconductors-101-part-5-the-mobile-ap-all-in-one/

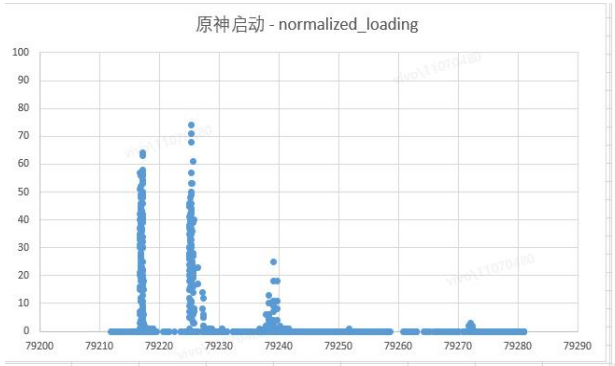
优化探索 - GPU异构压缩

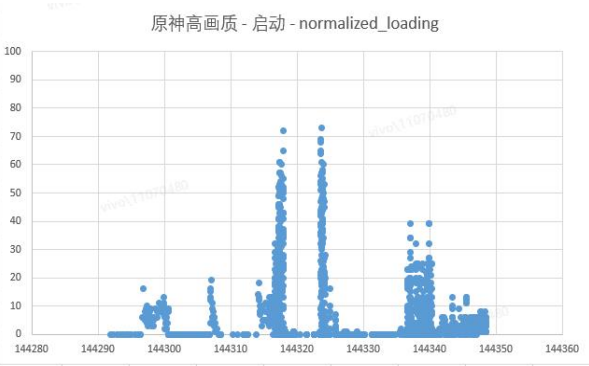
➢ GPU擅长处理大规模并行计算。而内存压缩，是对大量独立内存页进行高度并行的、模式固定的操作。

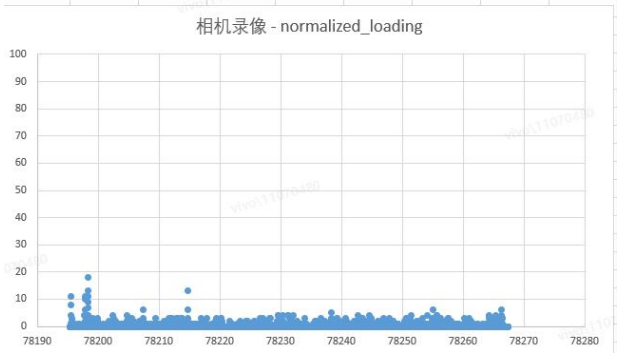
➢ 系统中大部分场景GPU负载不高：

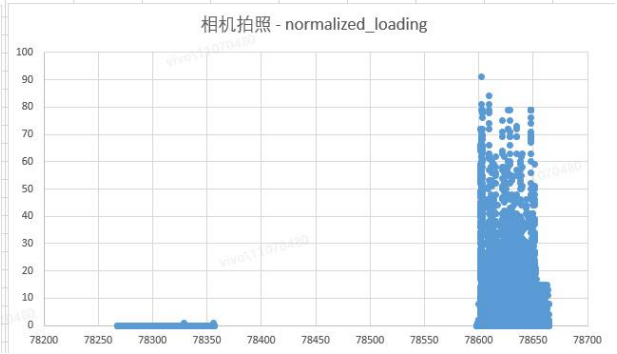


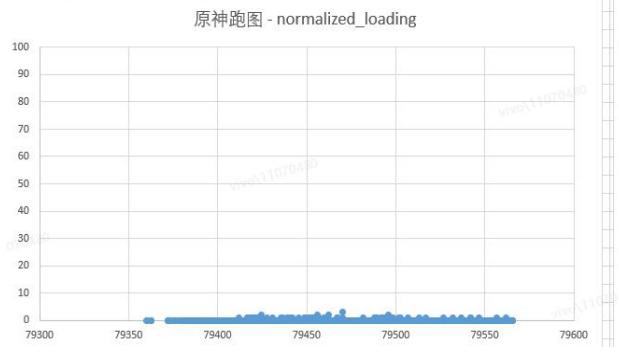


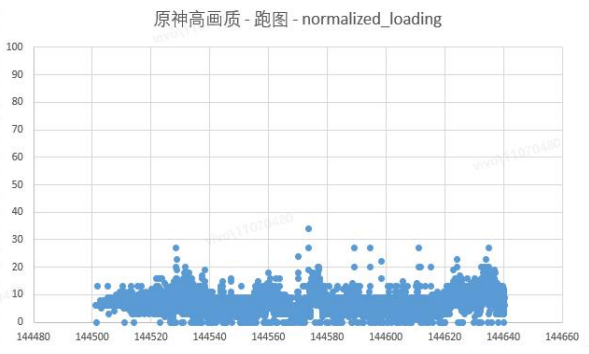




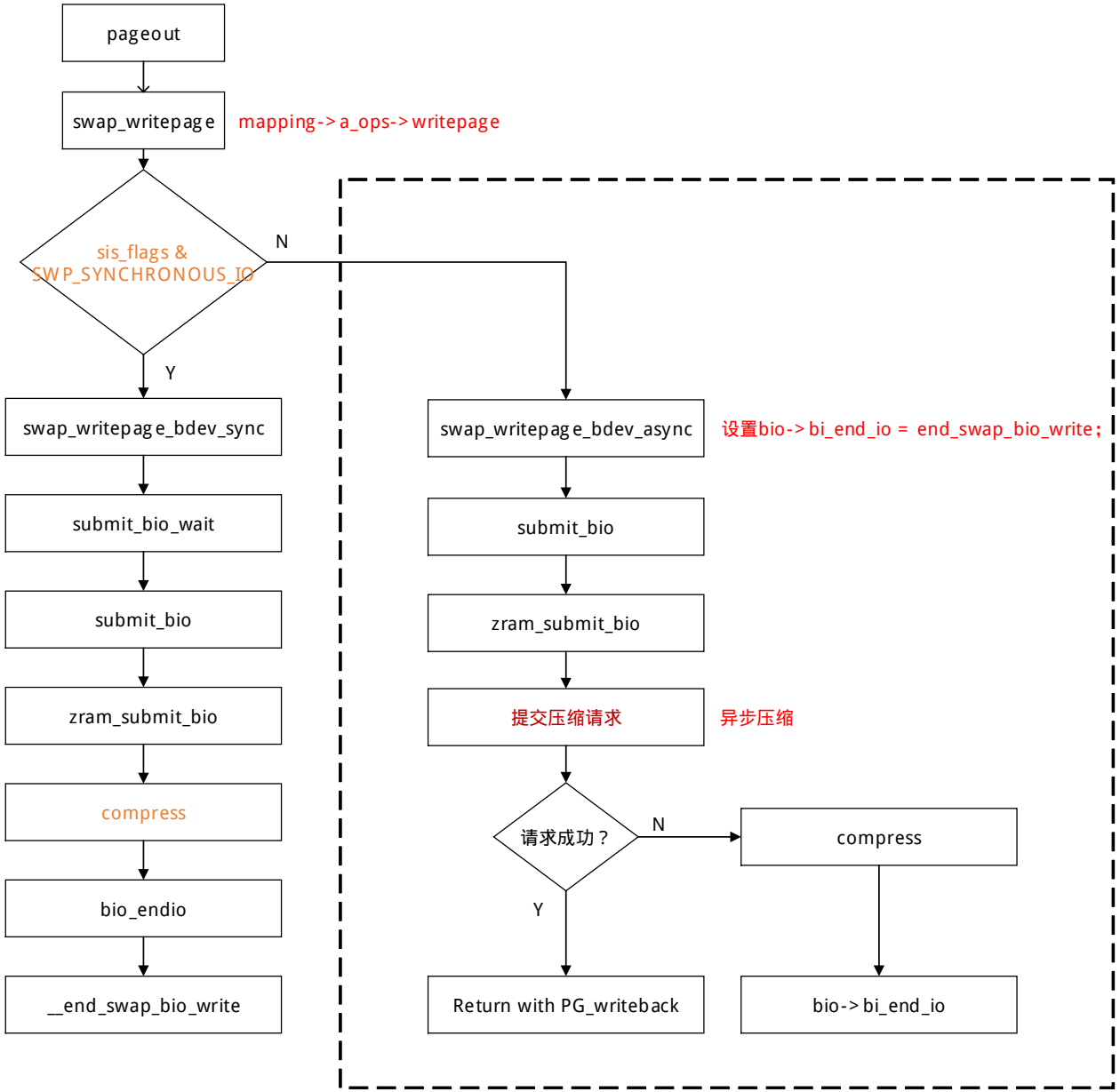








GPU异构压缩 - 异步压缩



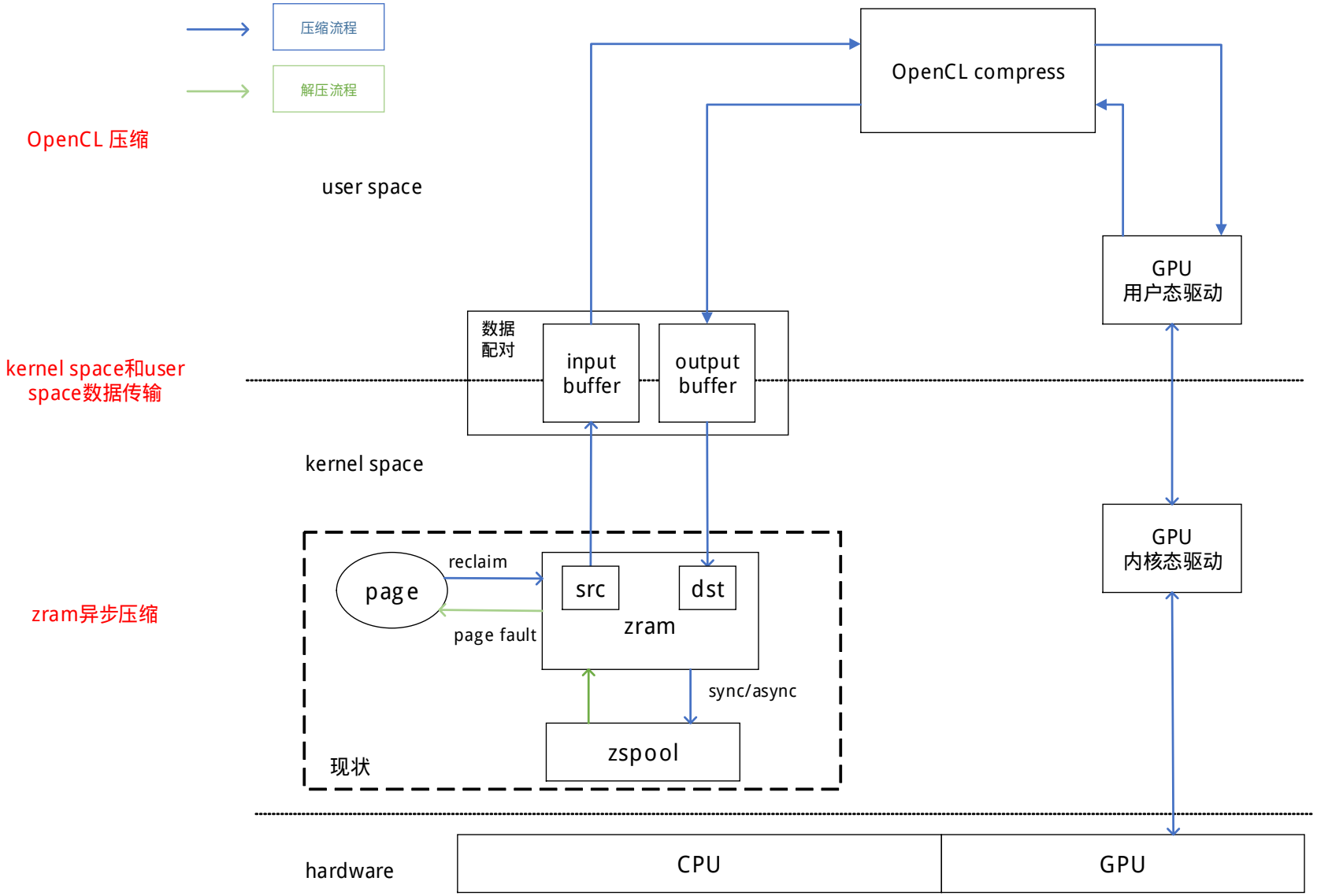
同步压缩：

- 目前匿名页回收通过swap_writepage_bdev_sync ->submit_bio_wait提交io请求到zram驱动
- zram驱动同步压缩页面
- __end_swap_bio_write做io结束之后的一些处理，如清理writeback标志等

异步压缩：

- swap_writepage_bdev_async设置bio_end_io回调，通过submit_bio提交io请求到zram驱动
- zram为页面提交压缩请求，请求不成功，则再做同步压缩，同步压缩之后通过bio_end_io回调做如writeback标志清理操作
- 异步压缩请求提交成功之后，带着writeback返回。待到压缩被真正执行结束之后，再通过bio_end_io回调进行清理。

GPU异构压缩 - 整体架构



1. **任务发起：**回收匿名页时，zram不再直接调用CPU压缩，而只是准备好压缩数据、用于存放压缩结果、压缩状态的内存缓冲区。

2. **跨态传输：**zram将这些数据通过特定的方式从内核态传递到用户态的一个OpenCL管理程序，我们称之为Host程序。

3. **GPU任务提交与执行：**OpenCL Host程序会负责将压缩需要参数传递给GPU可执行程序（kernel）。这个程序会经过GPU的用户态、内核态驱动等一系列复杂的流程，最终被调度到GPU硬件上开始执行压缩计算。

4. **结果回传：**压缩完之后再将压缩结果、压缩状态（成功或失败）回传给zram驱动。

5. **数据存储：**zram将压缩结果存储到zspool。

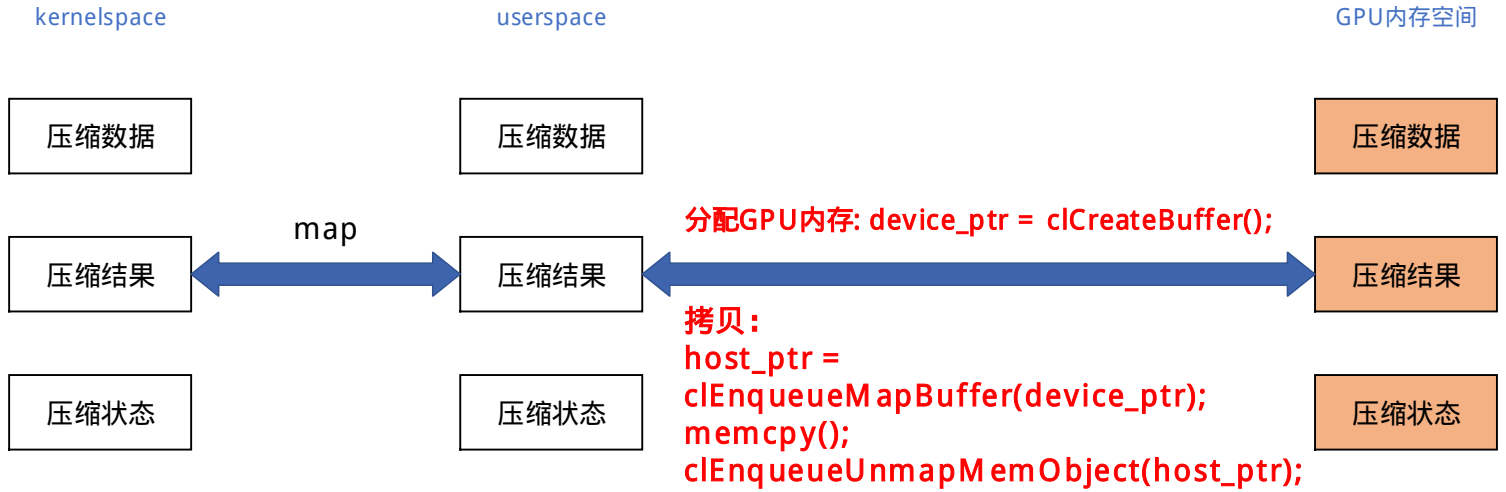
GPU异构压缩 - 批量数据管理

- 如果每次只提交一个页面给GPU压缩，GPU任务提交开销占比过高。
- 每一次GPU调用，无论数据量多小，都需要走过一个复杂的软件栈。从我们的驱动，到OpenCL，再到GPU内核驱动，最后才到达硬件。这个过程涉及多次上下文切换、内存管理、以及硬件命令提交等。
- 通过处理更大规模的数据来摊薄单次操作的平均成本，提升GPU的吞吐量。

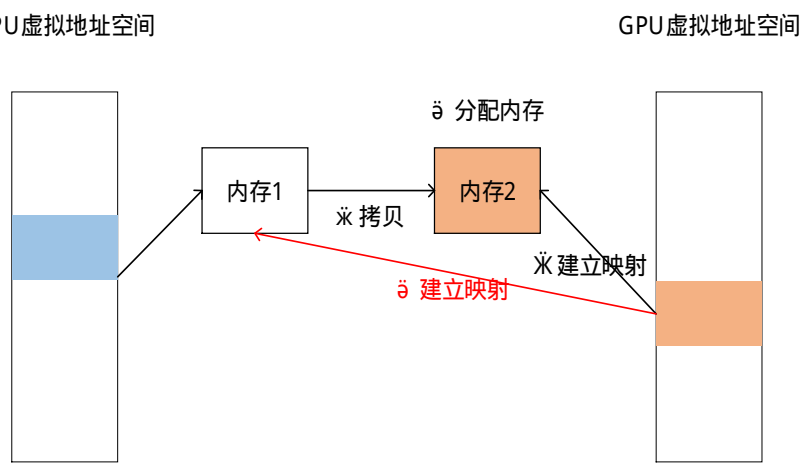
1. 独立的异步回收线程负责收集需要压缩的匿名页
2. 通过FIFO管理压缩数据，当FIFO填满时进行批量数据唤醒GPU压缩线程
3. GPU压缩线程传输数据给OpenCL，在压缩完成之后，拷贝压缩结果到zspool中

GPU异构压缩 - CPU-GPU零拷贝通道

➢ 数据流向



➢ CPU-GPU零拷贝



➢ 假设一次性压缩64MB内存

压缩数据 64MB

压缩结果 64MB/3≈21MB

压缩状态 (64MB/4KB)*8B=128KB

```
struct gpu_compress_desc {
        unsigned int comp_len;
        unsigned int status;
};
```

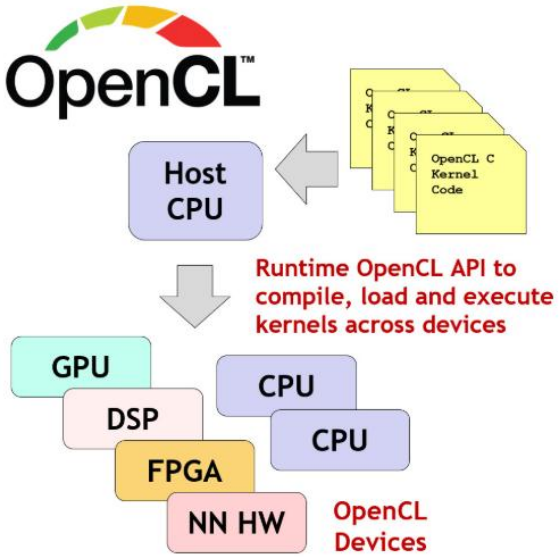
异构性能劣化:
1. 需要分配85MB+128KB内存
2. 每次压缩需要拷贝85MB+128KB内存

系统性能劣化:
1. CPU缓存污染
2. 内存带宽争抢

GPU异构压缩 - LZ4的GPU实践

➢ 使用OpenCL进行GPU编程

- Host（主机端）：运行在CPU上的主控程序，负责管理内存、准备数据并向GPU提交计算任务。
- Device（设备端）：即GPU硬件。Host提交的计算任务在Device上执行，这些任务被封装为Kernel ——一种可在GPU上并行执行的特定算法程序。

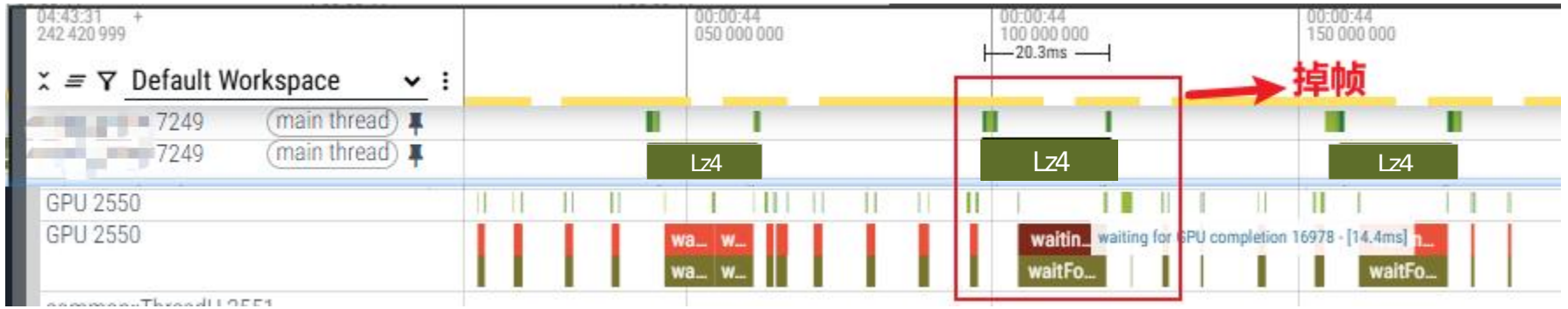


➢ 将内核LZ4算法直接移植为OpenCL kernel程序，实现一个GPU线程处理一个4KB内存页。

➢ 将内核完全串行的压缩算法改造成并行的，实现多个GPU线程协同压缩一个页面，性能比单线程压缩提高40%。

GPU异构压缩 - 计算和渲染冲突

➢ 当用户滑动屏幕时，如果因为GPU正忙于压缩内存，导致渲染掉帧，这是绝对不能容忍的：



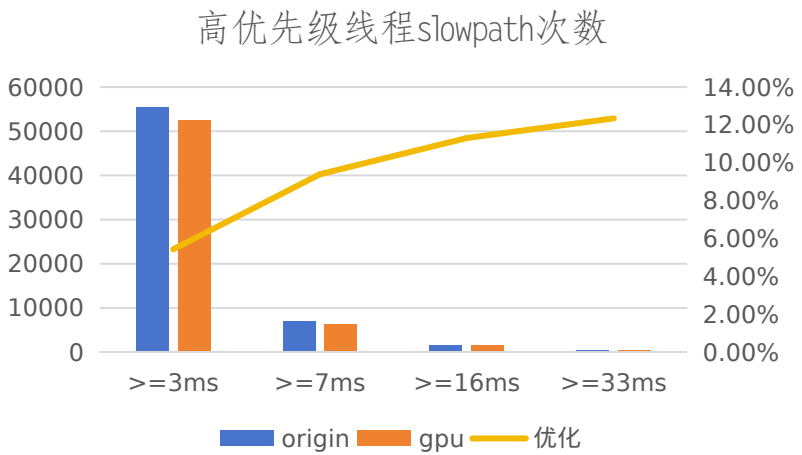
➢ GPU 的调度与 CPU 方式不同，GPU 虽然有多个执行单元，但同时间只能执行一个任务，通过排队“抢占”实现多任务切换。

1. **资源限额：**限制任务排队可用计算单元数量，为高优先级的渲染任务预留充足的硬件资源。

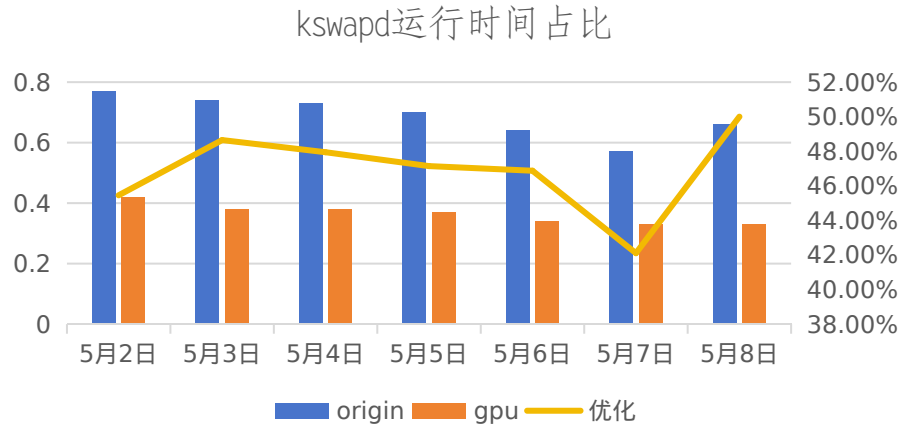
2. **任务拆分：**将kernel拆分成不同的小的kernel执行，每个kernel执行LZ4算法的一部分，使各个kernel执行得更快，调度器抢占颗粒度更小，速度更快。

3. **主动让出：**支持中停机制，当系统判断有重载渲染任务时，即使GPU正在执行压缩任务，也可以让压缩任务立即结束，让出GPU。

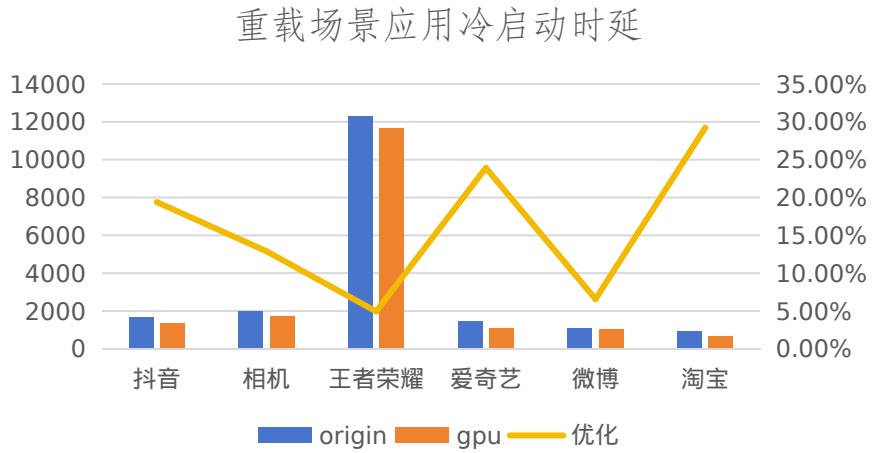
优化效果



- 高优先级线程slowpath次数降低5%-12%



- kswapd运行时间降低45%~50%



- 重载场景应用冷启动时延平均降低16.16%

展望

内核还有哪些适合在异构单元上执行的任务？GPU、NPU、专用的硬件芯片

THANKS