## Gen5 规格描述

Gen 5 容量 7.68T，128k 顺序写带宽 9GB/s。

| **容量** | **3.84T** | **7.68T** |
| --- | --- | --- |
| 128k 顺序读带宽 | 14 GB/s | 14 GB/s |
| 128k 顺序写带宽 | 5.5 GB/s | 9 GB/s |
| 4k 随机读IOPS | 2600k IOPS | 2600k IOPS |
| 4k 随机写IOPS | 215k IOPS | 350k IOPS |
| 4k 7/3随机读写IOPS | 500k | 800k |
| 4k 随机读延迟 | 66us | 66us |
| 4k 随机写延迟 | 12us | 12us |
| 4k 随机读99.99% Qos | 160us | 160us |
| 4k 随机写99.99% Q0s | 120us | 120us |
| 写寿命 | 1DWPD @5年 | 1 DWPD @5年 |
| 最大平均功耗 | 16W | 22W |

## 问题现象

fio 128k 顺序写带宽 5GB/s，仅达到 Gen5 规格的 50% 左右。

lat avg：3169us，iops： 40k，bw：5GB/s

fio --direct=1 --end\_fsync=1 --refill\_buffers --norandommap --randrepeat=0 --group\_reporting --log\_avg\_msec=5000 --name=fio-test --filename=/dev/nvme1n1 --numjobs=1 --rw=write --bs=128kb --ioengine=libaio --offset=0G --iodepth=128 --time\_based --runtime=600

## 问题分析

### fio 进程 CPU 使用率 100%

其中，%user 占比 ～90。重新编译 fio，带上 debug 信息，使用雨滴分析 fio 进程的热点函数。

[root@VM-1-8-tencentos ~]# pidstat -C fio -u 1

Linux 5.4.119-19-0009.11 (VM-1-8-tencentos) 02/05/2024 -×86\_64\_ (512 CPU)

07:31:00 PM UID PID %usr system %guest %wait %CPU CPU Command

07:31:01 PM 804700 88.79 11.21 0.00 0.00 100.00 36 fio

07:31:01 PM UID PID %usr system %guest %wait %CPU CPU Command

07:31:02 PM 804700 91.00 9.00 0.00 0.00 100.00 36 fio

07:31:02 PM UID PID %usr system %guest %wait %CPU CPU Command

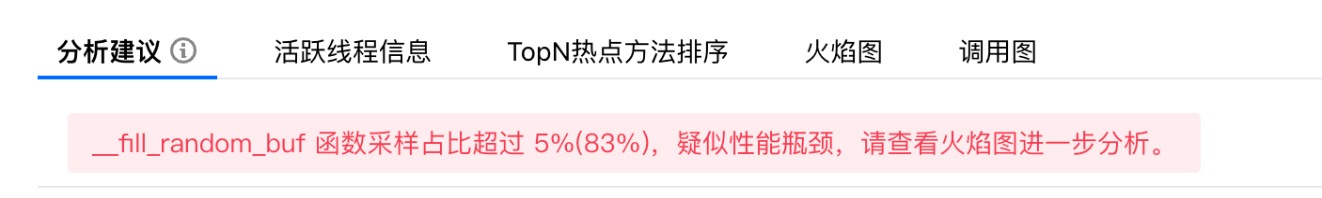
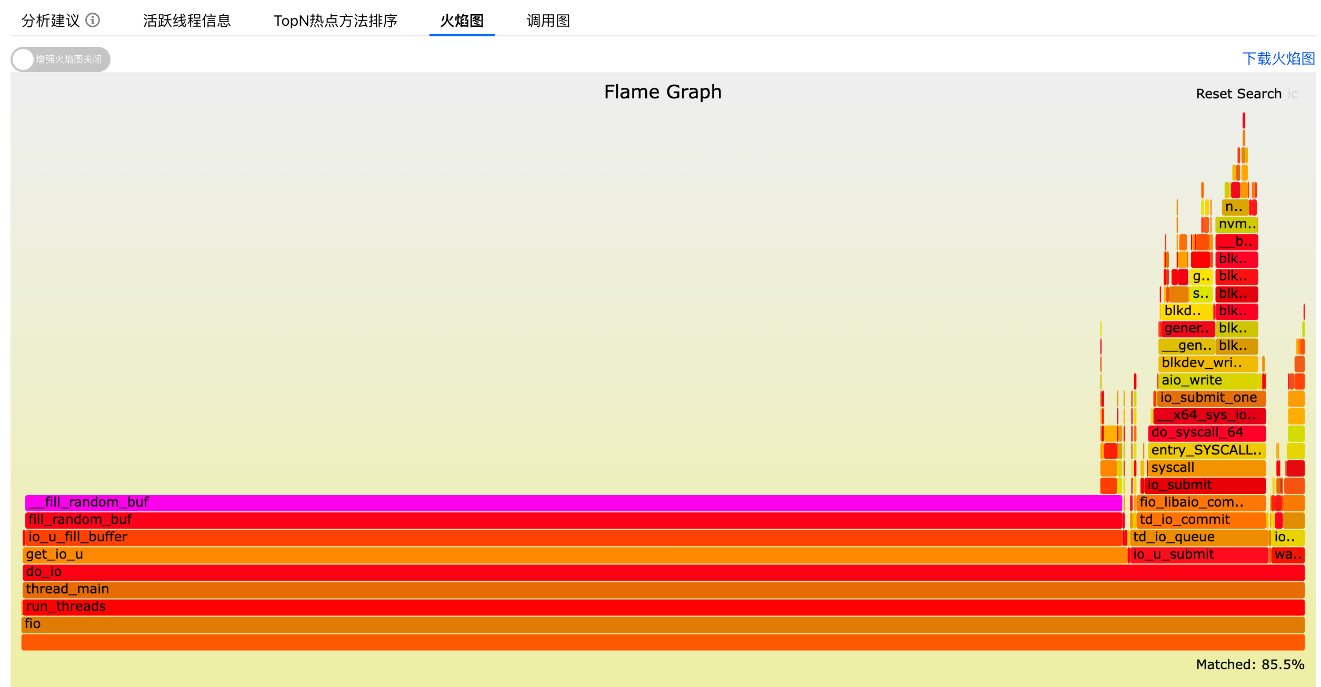
07:31:03 PM 804700 92.00 8.00 0.00 0.00 100.00 36 fio

07:31:03 PM UID PID %usr system %guest %wait %CPU CPU Command

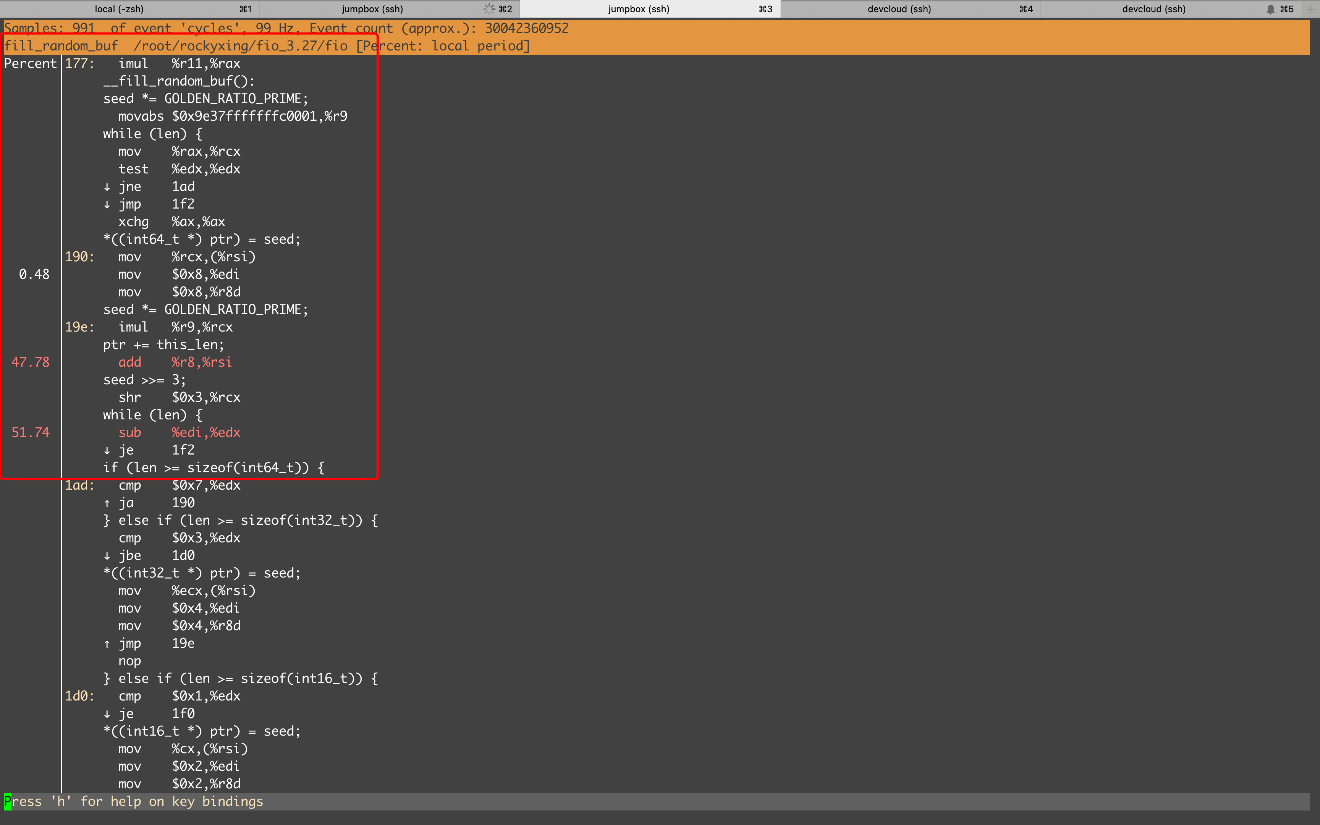
07:31:04 PM 804700 90.00 10.00 0.00 0.00 100.00 36 fio

### 雨滴任务结果

任务结果链接：[Drop - 一站式性能优化平台](https://drop.qcloud.com/task/result?taskID=743eecfd-45d2-4752-95a0-3f4578f957c6&taskExpiredTime=2022566715&sharedTaskToken=WUd7206715&isSharedURL=true)分析建议：

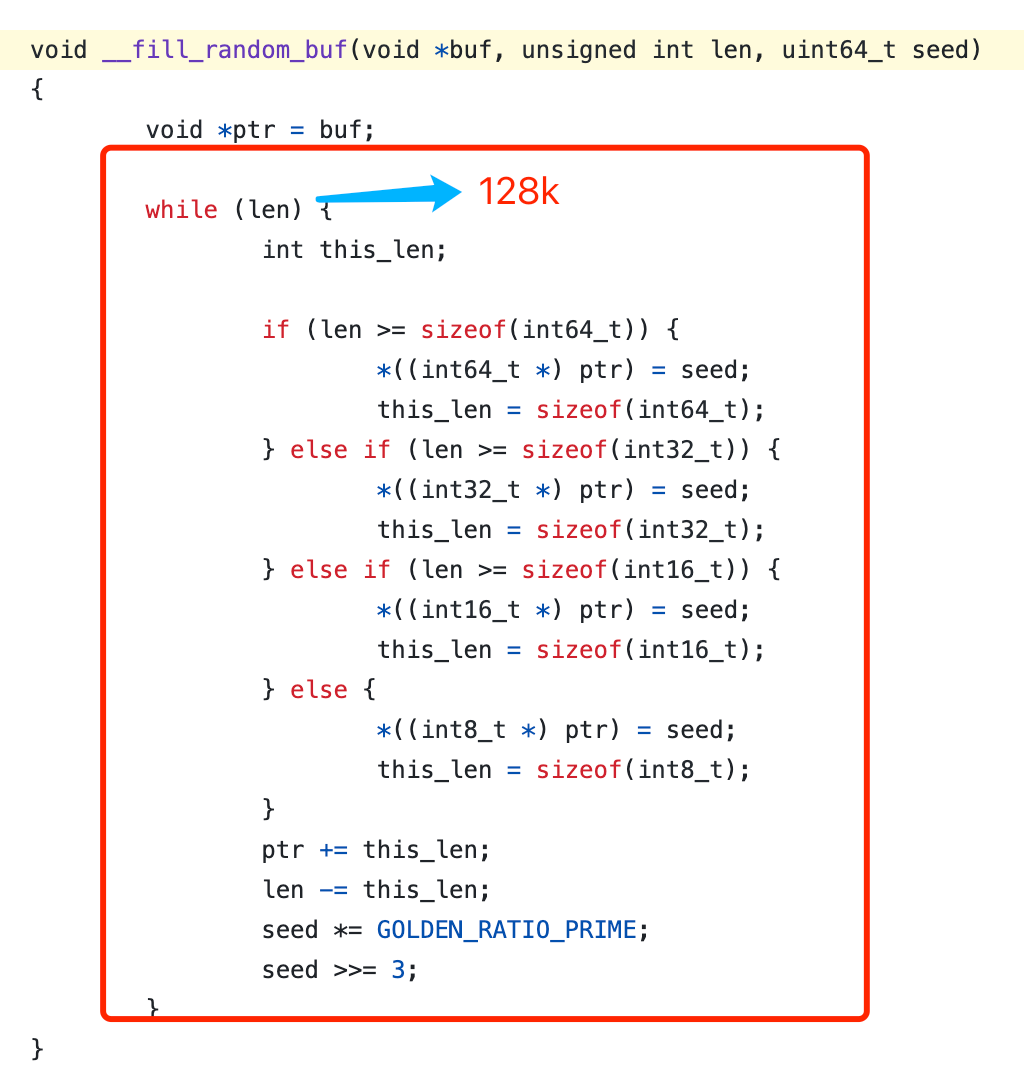
热点函数 fill\_random\_buf 占比 83%  
  


### 热点汇编 fill\_random\_buf

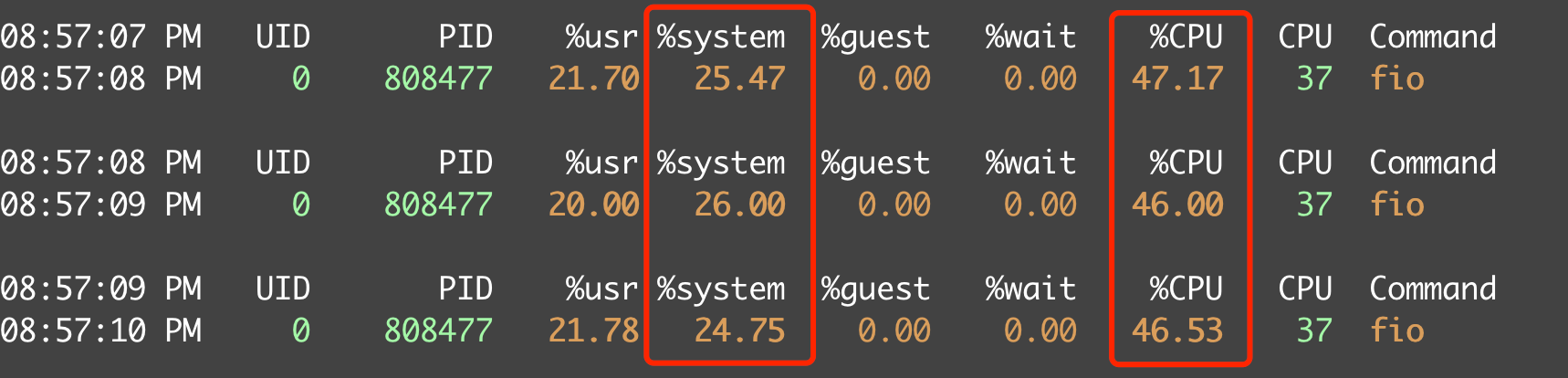
超过 99% 的 CPU 时间在 while 循环，其中随机填充 buffer 相关的累加、位移等操作是热点指令。  


### 源码分析

<https://github.com/axboe/fio/blob/fio-3.14/io_u.c#L1723>   
加了 refill\_buffer 参数，最终调到了 fill\_random\_buf： io\_u\_fill\_buffer -> fill\_io\_buffer -> fill\_random\_buf

fill\_random\_buf 源码如下：<https://github.com/axboe/fio/blob/fio-3.14/lib/rand.c#L98>   
  
开启 refill\_buffer，每次提交数据前 fio 都需要填充，里面又是一个大循环，涉及乘法、位移等计算，每秒钟至少增加 1 亿次（40K \* 128K \* 2）计算量，大约耗费 0.9 个核心，剩下 0.1 核心用于做数据提交。

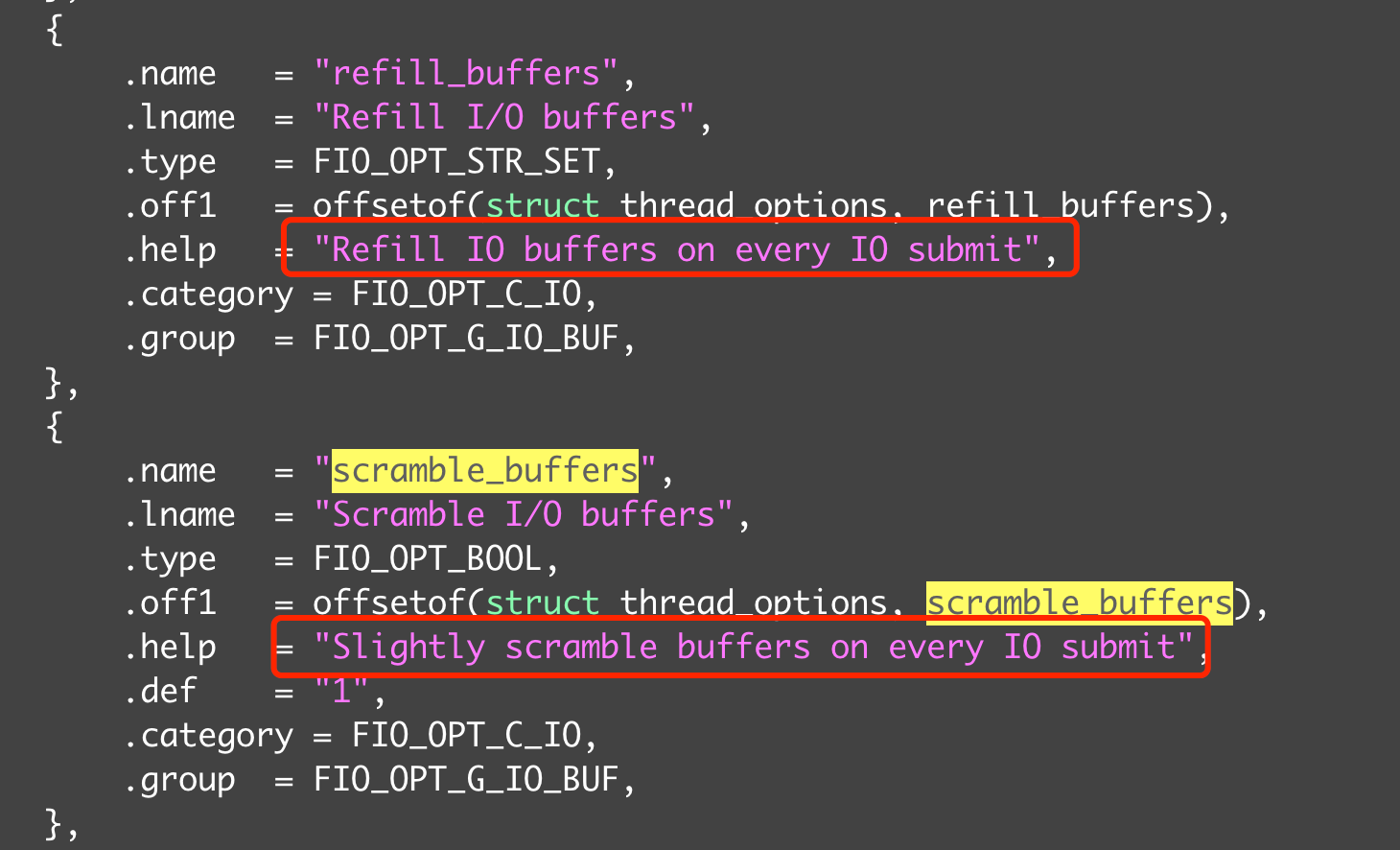
### 开启 vs 关闭 refill\_buffers

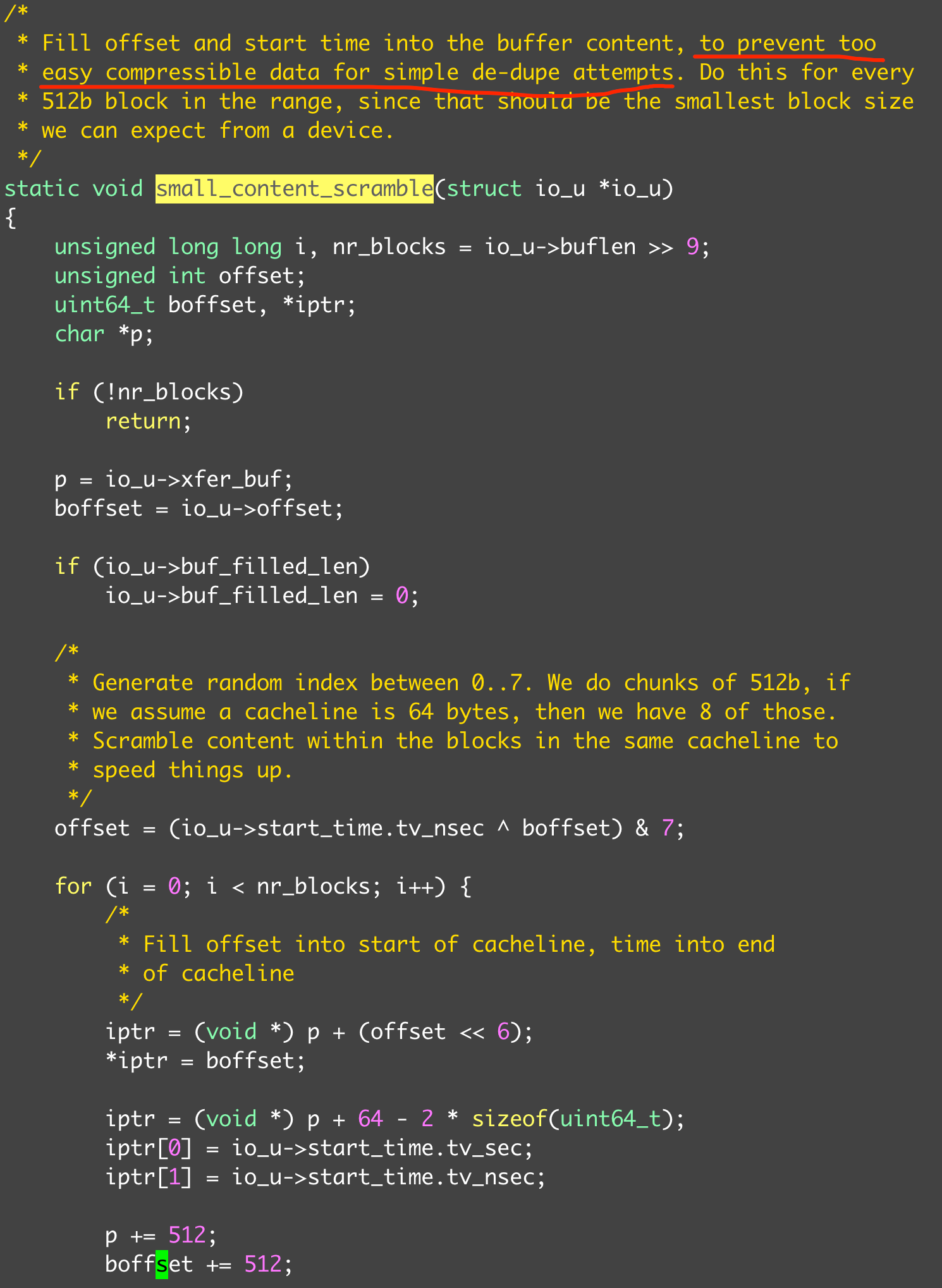
开启 refill\_buffer 时，fio 进程 CPU 使用率 100%，其中 %usr 占比超过 90%。  
  
关闭时 CPU 开销不到 50%  
  
每次提交 io 之前 refill buffer，拖慢了下一次的 io，性能就上不去了。

refill\_buffer=1，lat avg：3169us，iops 40k  
refill\_buffer=0，lat avg：1528us，iops 80k

## 解决方案

### 参数 scramble\_buffers 替代 refill\_buffers

  
scramble\_buffers 作用：轻微扰动下 buffer 内容，比 refill\_buffer 每个字节都重新计算填充代价应该小很多，也可以有效阻止数据压缩。

scramble\_buffers 原理：按每个 512 Byte 随机，注释也提到了防止容易数据压缩的情况（prevent too easy compressible data for simple de-dupe）  
  
开启参数 scramble\_buffers 时，CPU 使用率仅增加了 10% 左右。

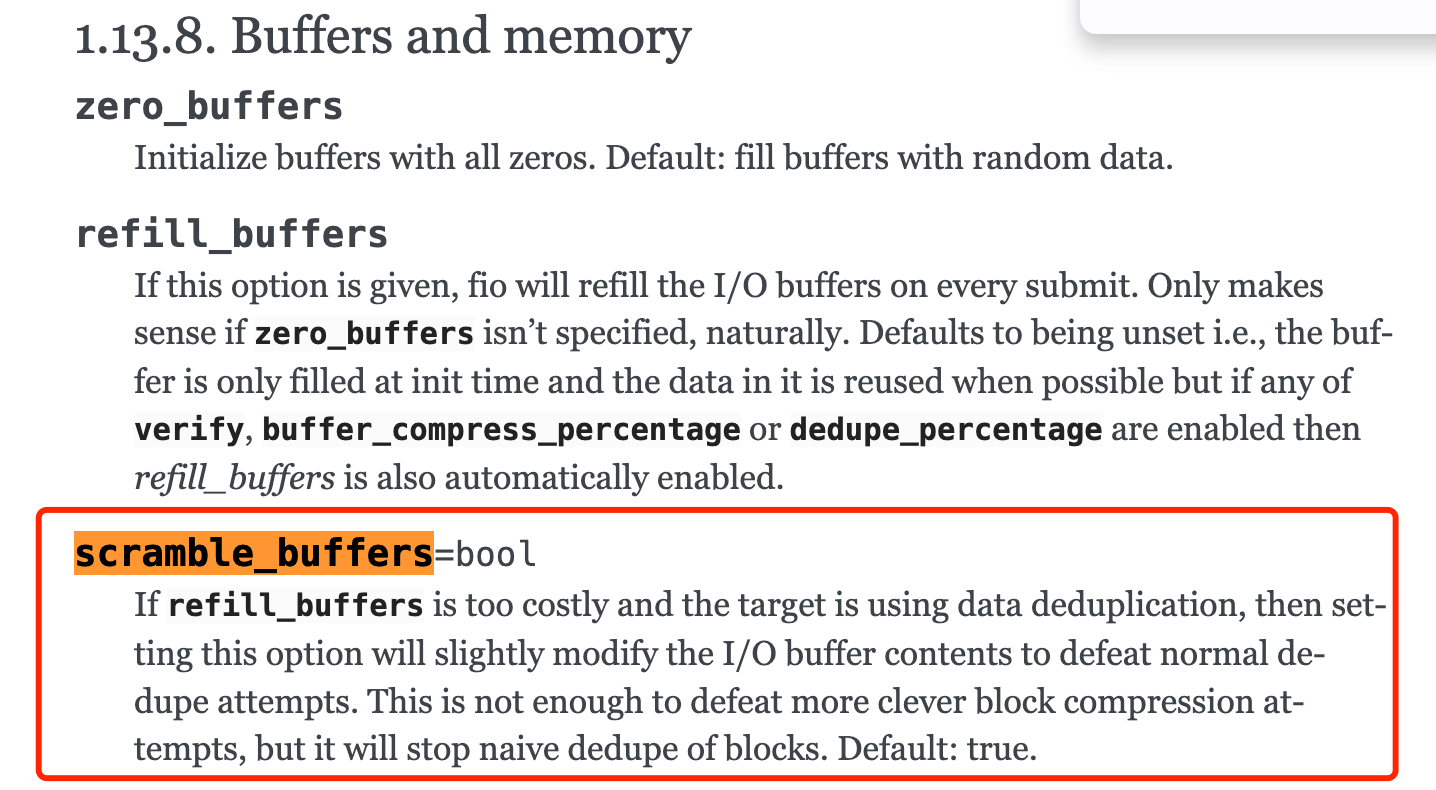
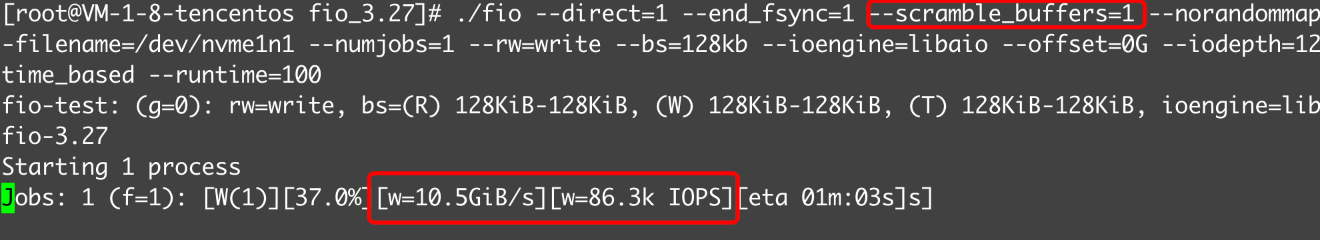
refill\_buffers=1： CPU 100%

scramble\_buffers=1： CPU ~60%

refill\_buffers=0 & scramble\_buffers=0: CPU ~50%

fio 文档也提到了这一点：

If refill\_buffers is too costly and the target is using data deduplication, then setting this option will slightly modify the I/O buffer contents to defeat normal de-dupe attempts. This is not enough to defeat more clever block compression attempts, but it will stop naive dedupe of blocks.

  
<https://fio.readthedocs.io/en/latest/fio_doc.html#cmdoption-arg-refill_buffers>   
采用参数 --scramble\_buffers 代替 --refill\_buffers，既可以保持提交数据一定程度的随机性，也可以把写带宽打满（10G/s)。  


## fio 相关源码

<https://fio.readthedocs.io/en/latest/fio_doc.html#cmdoption-arg-refill_buffers>   
<https://fio.readthedocs.io/en/latest/fio_doc.html#cmdoption-arg-scramble_buffers>   
<https://github.com/axboe/fio/blob/fio-3.14/io_u.c#L1723>   
<https://github.com/axboe/fio/blob/fio-3.14/lib/rand.c#L98>