

密级状态：绝密() 秘密() 内部() 公开(✓)

RK3399 性能优化方法

(技术部，第二系统产品部)

| | | |
|--|---------|------------|
| 文件状态： [] 正在修改 [✓] 正式发布 | 当前版本： | V1.1 |
| | 作 者： | 刘益星、郑建 |
| | 完成日期： | 2018-10-11 |
| | 审 核： | |
| | 完成日期： | |

福州瑞芯微电子股份有限公司

Fuzhou Rockchips Semiconductor Co., Ltd

(版本所有,翻版必究)

版 本 历 史

| 版本号 | 作者 | 修改日期 | 修改说明 | 备注 |
|------|-----|------------|--------------------------|----|
| V1.0 | 刘益星 | 2018-6-6 | 发布初始版本 | |
| V1.1 | 郑建 | 2018-10-11 | 方法三：增加指定线程绑定大核的demo 以及说明 | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

目 录

| | |
|--|---|
| 概述..... | 1 |
| 方法一：修改 interactive 调频策略的 target_loads..... | 1 |
| 方法二：为不同类别的任务分配 CPU 核资源..... | 2 |
| 方法三：指定线程绑定大核..... | 3 |
| 方法四：设置线程优先级..... | 4 |
| 方法五：提高 CCI 频率 | 8 |

概述

本文主要介绍针对 rk3399 android 系统的一些常用的性能优化方法。

方法一：修改 **interactive** 调频策略的 **target_loads**

RK3399 默认的调频策略是 interactive，此策略同时提供了一些参数供修改，其中最容易理解和修改的参数就是 target_loads，介绍如下：

Kernel/Documentation/cpu-freq/governors.txt:

2.6 Interactive

The CPUfreq governor "interactive" is designed for latency-sensitive, interactive workloads. This governor sets the CPU speed depending on usage, similar to "ondemand" and "conservative" governors, but with a different set of configurable behaviors.

The tuneable values for this governor are:

target_loads: CPU load values used to adjust speed to influence the current CPU load toward that value. In general, the lower the target load, the more often the governor will raise CPU speeds to bring load below the target. The format is a single target load, optionally followed by pairs of CPU speeds and CPU loads to target at or above those speeds. Colons can be used between the speeds and associated target loads for readability. For example:

```
85 1000000:90 1700000:99
```

targets CPU load 85% below speed 1GHz, 90% at or above 1GHz, until 1.7GHz

and above, at which load 99% is targeted. If speeds are specified these must appear in ascending order. Higher target load values are typically specified for higher speeds, that is, target load values also usually appear in an ascending order. The default is target load 90% for all speeds.

一般情况下，调速器根据 target_loads 参数调整频率，负载超过设定值时提高频率，反之则下降频率；该值设置的越低，CPU 越容易提升频率；单位: %，频率单位: KHz。

RK3399 mid SDK 默认的 target_loads 是 90，行业 sdk 默认的是 65。

修改 target_loads 方法，如下红色字体：

在 device/rockchip/rk3399/init.tablet.rc 中：

on boot

update cpusets feature nodes for rk3399 tablet

write /dev/cpuset/foreground/cpus 0-5

write /dev/cpuset/foreground/boost/cpus 4-5

write /dev/cpuset/background/cpus 0

write /dev/cpuset/system-background/cpus 0-3

write /dev/cpuset/top-app/cpus 0-5

write /sys/devices/system/cpu/cpufreq/policy4/interactive/target_loads 65

临时验证的话，手动修改 target_loads 的方法如下：

1) su

2) echo 65 >

/sys/devices/system/cpu/cpufreq/policy4/interactive/target_loads

方法二：为不同类别的任务分配 CPU 核资源

如上 init.tablet.rc 看到的，可以通过 linux 系统的 cpuset 子系统为不同任务分配 RK3399 的大小核资源（0 到 3 为小核，4 到 5 为大核）：

以 rk3399 行业 sdk 为例：

在 device/rockchip/rk3399/init.tablet.rc 中：

on boot

update cpusets feature nodes for rk3399 tablet

```
write /dev/cpuset/foreground/cpus 0-5
write /dev/cpuset/foreground/boost/cpus 4-5
write /dev/cpuset/background/cpus 0
write /dev/cpuset/system-background/cpus 0-3
write /dev/cpuset/top-app/cpus 0-5
write /sys/devices/system/cpu/cpufreq/policy4/interactive/target_loads 65
```

方法三：指定线程绑定大核

方法二中我们看到，android 上面可以把一类的进程指定到特定的 CPU 上运行以达到更优的运行效果，实现该功能的底层 API 函数其实是 sched_setaffinity(pid_t pid, unsigned int cpusetsize, cpu_set_t *mask)。

该函数设置进程为 pid 的这个进程，让它运行在 mask 所设定的 CPU 上；如果 pid 的值为 0，则表示指定的是当前进程，使当前进程运行在 mask 所设定的那些 CPU 上；第二个参数 cpusetsize 是 mask 所指定的数的长度，通常设定为 sizeof(cpu_set_t)；如果当前 pid 所指定的进程此时没有运行在 mask 所指定的任意一个 CPU 上，则该指定的进程会从其它 CPU 上迁移到 mask 的指定的一个 CPU 上运行。

这里提供一段代码示例：

```
cpu_set_t mask;
CPU_ZERO(&mask);
CPU_SET(4, &mask); //0 到 3 为小核，4 到 5 为大核
CPU_SET(5, &mask);
sched_setaffinity(0, sizeof(mask), &mask); //第一个参数是 pid，如果为 0 表示为当前进程
```

网上实例：https://blog.csdn.net/stn_lcd/article/details/78134574

命令行方式：

```
rk3399_mid:/ # taskset -p 674 //查看进程 674 分配的 CPU 核资源情况
pid 674's current affinity mask: 3f //3f 表示该进程可以跑在 cpu 0 到 5
rk3399_mid:/ # taskset -p 30 674 //绑定线程 674 到大核上 (cpu4,5)
pid 674's current affinity mask: 3f
```

pid 674's new affinity mask: 30

0 到 3 为小核，4 到 5 为大核，大核处理的性能效率更高，建议对 cpu 要求高的线程绑定到 4/5 核上，只绑定进程中 cpu 占用率高的线程。但是不建议绑定整个进程到 cpu4/5 上，因为如果进程创建了多于 2 个以上的线程，并且线程间存在互斥等待的情况，反而会导致该进程效率更低，所以建议只绑定进程中 cpu 占用率高的线程。

这里提供实际应用的代码示例：

1、将某个线程绑定到指定 CPU 的 4、5 大核，详情见下面代码：

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sched.h>
#include <ctype.h>
#include <pthread.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <dirent.h>
#include <grp.h>
#include <inttypes.h>
#include <pwd.h>
#include <string.h>
#include <jni.h>
#include <time.h>
#include <poll.h>
#include <fcntl.h>
#include <android/log.h>
#include <sys/stat.h>
#include <sys/syscall.h>
#include <assert.h>
#define TAG "native-lib"

#define DEBUG 1
#ifdef DEBUG
#define LOGD(...) __android_log_print(ANDROID_LOG_DEBUG, TAG, __VA_ARGS__)
#define LOGE(...) __android_log_print(ANDROID_LOG_ERROR, TAG, __VA_ARGS__)
#else
#define LOGD(...) ((void)0)
#define LOGE(...) ((void)0)
#endif
```

```
void set_cur_thread_affinity(cpu_set_t *mask)
{
    int err, syscallres;
    pid_t tid = getpid();
    print_cpu_mask(*mask);
    syscallres = syscall(__NR_sched_setaffinity, tid,
sizeof(cpu_set_t), mask);
    if (syscallres) {
        err = errno;
        LOGE("Error in the syscall setaffinity: mask = %d,
err=%d", mask, errno);
    }
    LOGD("tid = %d has setted affinity success", tid);
}

JNIEXPORT void JNICALL Java_nativelibs_Affinity_bindThreadToCpu45(JNIEnv*
*env, jclass type)
{
    cpu_set_t mask;
    CPU_ZERO(&mask);
    CPU_SET(4, &mask);
    CPU_SET(5, &mask);
    set_cur_thread_affinity(&mask);
    LOGD("set affinity to %d success");
}
```

linux 下可以直接调用 `pthread_setaffinity_np`，将当前线程绑定在具体的 cpu 上，而 android 该 API 被屏蔽了，需要调用 `sched` 这个系统 API。

2、将整个进程绑定到指定 CPU 的 4、5 大核

```
JNIEXPORT int JNICALL Java_nativelibs_Affinity_bindProcessToCpu45(JNIEnv*
env, jobject obj) {
    DIR *task_dir;
    struct dirent *tid_dir;
    int pid, tid;
    int ret;
    char filename[64];
    LOGD("BindThreadsToCpu45 : Current platform = rk3399");
    cpu_set_t mask;
    struct sched_param param;
    CPU_ZERO(&mask);
    CPU_SET(4, &mask);
    CPU_SET(5, &mask);
    extern int errno;
    // bind whole processId to cpu 4 & 5
```



```
pid = getpid();
LOGD("pid = %d", pid);
ret = sched_setaffinity(pid, sizeof(mask), &mask);
if(ret < 0) {
    LOGE("BindThreadsToCpu45 -> bind process sched_setaffinity
return %d, errno = %d\n", ret, errno);
    return -1;
} else {
    LOGD("BindThreadsToCpu45 -> bind process sched_setaffinity
success!");
}
return 0;
}
```

方法四：设置线程优先级

POSIX 标准指定了三种调度策略：先入先出策略 (SCHED_FIFO)、循环策略 (SCHED_RR) 和自定义策略 (SCHED_OTHER)。SCHED_FIFO 和 SCHED_RR 属于 RT 线程（实时线程），Android 有很严格的系统权限限制，上层的线程是无法设置为 RT 线程，可设置为普通线程中的高优先级线程，普通线程的最高优先级线程接近于 RT 线程。Android 线程优先级设置方法下面也会介绍。

API 函数：int sched_setscheduler(pid_t pid, int policy,

const struct sched_param *param);

函数将 pid 所指定进程的调度策略和调度参数分别设置为 policy 和 param 指向的 sched_param 结构中指定的参数。sched_param 结构中的 sched_priority 成员的值可以为任何整数，该整数位于 policy 所指定调度策略的优先级范围内(含边界值)。policy 参数的可能值在头文件<sched.h>中定义。

简单例子：

```
struct sched_param param = {
    .sched_priority = 90,
};

sched_setscheduler(0, SCHED_FIFO, &param); //第一个参数为 pid, 0 表示为当前进程
```

网上实例：<https://blog.csdn.net/allwtg/article/details/5254306>

命令行方式:

```
rk3399_mid:/ # busybox chrt -p 674
pid 674's current scheduling policy: SCHED_OTHER
pid 674's current scheduling priority: 0
1|rk3399_mid:/ # busybox chrt -f -p 90 674
pid 674's current scheduling policy: SCHED_OTHER
pid 674's current scheduling priority: 0
pid 674's new scheduling policy: SCHED_FIFO
pid 674's new scheduling priority: 90
```

android 系统在进程创建的时候实际上也有不同的优先级可以指定，frameworks/base/core/java/android/os/Process.java 中定义了下面这些优先级：

```
public static final int THREAD_PRIORITY_DEFAULT = 0; 应用的默认优先级
public static final int THREAD_PRIORITY_LOWEST = 19; 线程的最低优先级
public static final int THREAD_PRIORITY_BACKGROUND = 10; 后台线程的默认优先级

public static final int THREAD_PRIORITY_FOREGROUND = -2; 前台进程的标准优先级

public static final int THREAD_PRIORITY_DISPLAY = -4; 系统用于显示功能的优先级

public static final int THREAD_PRIORITY_URGENT_DISPLAY = -8; 系统用于重要显示功能的优先级

public static final int THREAD_PRIORITY_AUDIO = -16; 音频线程默认优先级
public static final int THREAD_PRIORITY_URGENT_AUDIO = -19; 重要音频线程默认优先级
```

这些优先级的设定，通过层层调用，实际上最终还是由 sched_setscheduler 执行。

简单的例子（如下红色字体）：

```
@@ -3,6 +3,7 @@ package com.google.vrtoolkit.cardboard.sensors;
import android.hardware.*;
import java.util.*;
import android.os.*;
```

```

+import android.os.Process;

import android.util.*;

public class DeviceSensorLooper implements SensorEventProvider
@@ -49,7 +50,7 @@ public class DeviceSensorLooper implements
SensorEventProvider
    }
}
};

-    final HandlerThread sensorThread = new HandlerThread("sensor")
{
+    final HandlerThread sensorThread = new HandlerThread("sensor",
Process.THREAD_PRIORITY_URGENT_AUDIO) {
    protected void onLooperPrepared() {
        final Handler handler = new Handler(Looper.myLooper());
        final Sensor accelerometer =
DeviceSensorLooper.this.sensorManager.getDefaultSensor(1);

```

方法五：提高 CCI 频率

如果关注 ARM 的 DDR 吞吐率，可以提高 CCI 频率来提升，也就是说对于那些瓶颈在 memory bandwidth 的应用，需要把 cci 的频率提高到和 ddr 一样，例如，目前默认配置下 ddr 频率跑 800M, cci 频率是 300M, 把 cci 改成 800M 则可以显著提高 ddr 的吞吐率。

CCI 频率修改方法：

```

--- a/arch/arm64/boot/dts/rockchip/rk3399-vop-clk-set.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3399-vop-clk-set.dtsi
@@ -171,7 +171,7 @@

        <400000000>, <200000000>,
        <400000000>, <300000000>,
        <300000000>, <300000000>,

```

```
- <300000000>, <300000000>,
+ <300000000>, <800000000>,
  <1000000000>, <1500000000>,
  <1500000000>, <4000000000>,
  <1000000000>, <4000000000>,
```

对应关系:

```
rk3399 Project - Source Insight - [Rk3399-vop-clk-set.dtsi (arch/arm64/boot/dts/rockchip)]
File Edit Search Project Options View Window Help
<&cru SCLK_I2C2>, <&cru SCLK_I2C3>,
<&cru SCLK_I2C5>, <&cru SCLK_I2C6>,
<&cru SCLK_I2C7>, <&cru SCLK_SPI0>,
<&cru SCLK_SPI1>, <&cru SCLK_SPI2>,
<&cru SCLK_SPI4>, <&cru SCLK_SPI5>,
<&cru ACLK_GIC>, <&cru ACLK_ISP0>,
<&cru ACLK_ISP1>, <&cru SCLK_VOP0_PWM>,
<&cru SCLK_VOP1_PWM>, <&cru PCLK_EDP>,
<&cru ACLK_HDCP>, <&cru ACLK_VIO>,
<&cru HCLK_SD>, <&cru SCLK_CRYPT00>,
<&cru SCLK_CRYPT01>, <&cru SCLK_EMMC>,
<&cru ACLK_EMMC>, <&cru ACLK_CENTER>,
<&cru ACLK_IEP>, <&cru ACLK_RGA>,
<&cru SCLK_RGA_CORE>, <&cru ACLK_VDU>,
<&cru ACLK_VCODEC>, <&cru PCLK_DDR>,
<&cru ACLK_GMAC>, <&cru SCLK_VDU_CA>,
<&cru SCLK_VDU_CORE>, <&cru ACLK_USB3>,
<&cru FCLK_CM0S>, <&cru ACLK_CCI>,
<&cru PCLK_ALIVE>, <&cru SCLK_CS>,
<&cru SCLK_CCI_TRACE>, <&cru ACLK_VOP0>,
<&cru HCLK_VOP0>, <&cru ACLK_VOP1>,
<&cru HCLK_VOP1>;
assigned-clock-rates =
<75000000> <50000000>
```

```
rk3399 Project - Source Insight - [Rk3399-vop-clk-set.dtsi (arch/arm64/boot/dts/rockchip)]
File Edit Search Project Options View Window Help
<500000000>, <1000000000>,
<500000000>, <1000000000>,
<1000000000>, <1000000000>,
<1000000000>, <1000000000>,
<1000000000>, <500000000>,
<500000000>, <500000000>,
<500000000>, <500000000>,
<2000000000>, <4000000000>,
<4000000000>, <1000000000>,
<1000000000>, <1000000000>,
<4000000000>, <4000000000>,
<2000000000>, <1000000000>,
<2000000000>, <2000000000>,
<1000000000>, <4000000000>,
<4000000000>, <4000000000>,
<4000000000>, <3000000000>,
<4000000000>, <2000000000>,
<4000000000>, <3000000000>,
<3000000000>, <3000000000>,
<3000000000>, <800000000>,
<1000000000>, <1500000000>,
<1500000000>, <4000000000>,
<1000000000>, <4000000000>,
<1000000000>;
};
#endif
```