Perf使用说明

发布版本：1.0

作者邮箱：cmc@rock-chips.com

日期：2017.12

文件密级：公开资料

前言

概述

产品版本

芯片名称	内核版本
全系列	4.4

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

日期	版本	作者	修改说明
2017-12-25	V1.0	陈谋春	

---

1 介绍

Perf是从Linux 2.6开始引入的一个profiling工具，通过访问包括pmu在内的软硬件性能计数器来分析性能，支持多架构，是目前Kernel的主要性能检测手段，和Kernel代码一起发布，所以兼容性良好。

2 功能

性能瓶颈如果要分类的话，大致可以分为几个大类：cpu / gpu / mem / storage，其中gpu用Perf没法探测（这个目前比较好用的工具就只有DS5），storage只能用tracepoint来统计。总的说来，Perf还是侧重于分析cpu的性能，其他功能都不是很好用。

```
$ perf

 usage: perf [--version] [--help] COMMAND [ARGS]

 The most commonly used perf commands are:
  annotate        Read perf.data (created by perf record) and display annotated code
  archive         Create archive with object files with build-ids found in perf.data file
  bench           General framework for benchmark suites
  buildid-cache   Manage <tt>build-id</tt> cache.
  buildid-list    List the buildids in a perf.data file
  diff            Read two perf.data files and display the differential profile
  inject          Filter to augment the events stream with additional information
  kmem            Tool to trace/measure kernel memory(slab) properties
  kvm             Tool to trace/measure kvm guest os
  list            List all symbolic event types
  lock            Analyze lock events
  probe           Define new dynamic tracepoints
  record          Run a command and record its profile into perf.data
  report          Read perf.data (created by perf record) and display the profile
  sched           Tool to trace/measure scheduler properties (latencies)
  script          Read perf.data (created by perf record) and display trace output
  stat            Run a command and gather performance counter statistics
  test            Runs sanity tests.
  timechart       Tool to visualize total system behavior during a workload
  top             System profiling tool.

 See 'perf help COMMAND' for more information on a specific command.
```

其中比较常用的功能有几个：

- record：收集profile数据
- report：根据profile数据生成统计报告
- stat：打印性能计数统计值
- top：cpu占有率实时统计

3 在Android平台使用

3.1 准备工作

1. 首先按Google或芯片厂商的指导，构建一个完整的Android和Kernel的编译环境（如果不关心Kernel可以忽略），这样分析的时候符号表才能匹配上。
2. 编译Perf

```
~$ . build/envsetup.sh
~$ lunch
~$ mmm external/linux-tools-perf
~$ adb root
~$ adb remount
~$ adb push perf /system/bin/
~$ adb shell sync

```

3. 准备符号文件

符号文件可以简单分为三类：

a. 平台native代码，这部分代码在编译的过程中会自动生成符号表，不需要我们干预

b. 平台java代码，对于art虚拟机来说（老版本的dalvik就不说了）最终的编译结果是oat文件，这也是正规的elf文件，但是默认是不带debug信息。而新版本的Android也提供了自动生成java符号表的工具：

```
bash art/tools/symbolize.sh
```

c. 第三方apk，如果是来自开源社区，则可以通过修改makefile和套用Android提供的java符号表工具来生成符号表文件，然后拷贝到Android的符号表目录，注意路径必须要和设备上的完全一致，可以通过showmap来获取设备上的路径。

```
~$ adb shell showmap apk_pid
38540    36296    36296        0        0    36216       80        0    3
/data/app/com.android.webview-2/lib/arm/libwebviewchromium.so
~$ cp libwebviewchromium.so $ANDROID_PRODUCT_OUT/symbols/data/app/com.android.webview-
2/lib/arm/libwebviewchromium.so
```

如果是商业的apk，基本上已经做过混淆和strip，除非开发商能配合，不然就没招。

4. 稍微新一点的Android都开起了Kernel的指针保护，这也会影响Perf的record，所以需要临时关闭保护：

```
~$ adb shell echo 0 > /proc/sys/kernel/kptr_restrict
```

5. 为了方便分析，一般会把record的数据pull到host端，在host端做分析，所以需要在设备端也安装一下Perf工具，ubuntu下安装命令如下：

```
~$ sudo apt-get install linux-tools-common
```

6. 目前大部分的Android平台默认Perf功能都是打开的，所以一般不需要重新配置Kernel，如果碰到Perf被关闭的情况，可以打开下面几个配置

```
CONFIG_PERF_EVENTS=y
CONFIG_HW_PERF_EVENTS=y
```

3.2 获取当前平台支持的事件

```
rk3399:/data/local # ./perf list



List of pre-defined events (to be used in -e):
  cpu-cycles OR cycles                               [Hardware event]
  instructions                                       [Hardware event]
  cache-references                                   [Hardware event]
  cache-misses                                       [Hardware event]
  branch-instructions OR branches                    [Hardware event]
  branch-misses                                      [Hardware event]
  bus-cycles                                         [Hardware event]

  cpu-clock                                          [Software event]
  task-clock                                         [Software event]
  page-faults OR faults                              [Software event]
  context-switches OR cs                             [Software event]
  cpu-migrations OR migrations                       [Software event]
  minor-faults                                       [Software event]
  major-faults                                       [Software event]
  alignment-faults                                   [Software event]
  emulation-faults                                   [Software event]
  dummy                                              [Software event]

  L1-dcache-loads                                    [Hardware cache event]
  L1-dcache-load-misses                              [Hardware cache event]
  L1-dcache-stores                                   [Hardware cache event]
  L1-dcache-store-misses                             [Hardware cache event]
  L1-dcache-prefetch-misses                          [Hardware cache event]
  L1-icache-loads                                    [Hardware cache event]
  L1-icache-load-misses                              [Hardware cache event]
  dTLB-load-misses                                   [Hardware cache event]
  dTLB-store-misses                                  [Hardware cache event]
  iTLB-load-misses                                   [Hardware cache event]
  branch-loads                                       [Hardware cache event]
  branch-load-misses                                 [Hardware cache event]
```

实际上Android移植的Perf还不完整，tracepoint的事件还不支持，例如：block事件，所以如果想要抓去一些内核子系统的性能信息就无法满足。Android 7.0开始已经去掉了Perf工具，替代它的是Simpleperf[1]工具，对tracepoint的支持比原来的好很多。

3.3 获取系统热点进程

Perf中的top工具可以列出当前cpu的热点，还可以附加Kernel的符号表让信息可方便分析。命令如下：

```
$ adb shell mkdir -p /data/local/symbols
$ adb push vmlinux /data/local/symbols/vmlinux
$ adb shell
# perf top --vmlinux=/path/to/vmlinux -d 2
```

结果输出如下：

```
PerfTop:    8272 irqs/sec  kernel:24.2%  exact:  0.0% [4000Hz cycles],  (all, 6 CPUs)
------------------------------------------------------------------------------------------

  62.47%  perf                 [.] 0x00000000001a3944
   2.34%  perf                 [.] strstr
   2.18%  [kernel]             [k] _raw_spin_unlock_irq
   2.03%  perf                 [.] strlen
   1.75%  perf                 [.] memcpy
   1.38%  [kernel]             [k] _raw_spin_unlock_irqrestore
   1.11%  [kernel]             [k] __compat_put_timespec
   1.03%  perf                 [.] je_malloc
   1.03%  perf                 [.] ifree
   0.88%  perf                 [.] strcmp
   0.81%  [kernel]             [k] el0_svc_naked
   0.78%  [kernel]             [k] cpuidle_enter_state
   0.67%  perf                 [.] pthread_getspecific
   0.56%  perf                 [.] je_free
   0.47%  [kernel]             [k] __arch_copy_to_user
```

perf top还可以只抓取指定进程的pid，这一般是用在要优化某个程序是非常有用，命令如下：

```
perf top --vmlinux=/path/to/vmlinux -d 2 -p pid_of_prog
```

perf top还和系统的top一样可以指定刷新间隔[2]，以上命令中的-d选项就是这个功能，单位是秒。

3.4 获取进程的统计信息

perf stat用于获取进程某个时间段内的pmu统计信息，命令如下：

```
 # ./perf stat -p 1415
```

ctrl+c退出，或发信号让Perf进程退出都可以看到统计结果，例如：

```
Performance counter stats for process id '1415':

     25802.685639 task-clock                #    2.010 CPUs utilized          [100.00%]
            28571 context-switches          #    0.001 M/sec                  [100.00%]
             3362 cpu-migrations            #    0.130 K/sec                  [100.00%]
              761 page-faults               #    0.029 K/sec
      42238237278 cycles                    #    1.637 GHz                    [64.17%]
   <not supported> stalled-cycles-frontend
   <not supported> stalled-cycles-backend
      15935073463 instructions              #    0.38  insns per cycle        [64.17%]
        713605132 branches                  #   27.656 M/sec                  [35.82%]
        262718809 branch-misses             #   36.82% of all branches        [64.18%]

     12.834800415 seconds time elapsed
```

一些明显的异常值会被标注为红色，例如上图是浏览器跑fishtank时候抓的统计信息，可以看到分支预测的失败率非常高，结合Perf的热点分析工具可以进一步缩小范围找到分支预测失败的原因。

3.5 收集进程的profile数据

perf record用于记录详细的profile数据，可以指定记录某个进程，还可以记录调用栈，命令如下：

```
# perf record -g -p pid -o /data/local/perf.data
```

也可以指定只抓取某个事件，事件列表可以通过上面的perf list得到，例如：

```
# ./perf record -e cache-misses -p 1415
```

3.6 分析profile数据

perf report用户分析抓到的profile数据，一般会先把数据发到pc上再分析，命令如下：

```
$ adb pull /data/local/perf.data
$ perf report --objdump=aarch64-linux-android-objdump --vmlinux=/path/to/vmlinux --symfs
$ANDROID_PRODUCT_OUT/symbols -i perf.data
```

结果如图：

```
Samples: 31K of event 'cycles', Event count (approx.): 12723245421
  Children      Self  Command          Shared Object                      Symbol
+    4.91%     0.00%  Chrome_InProcRe  [unknown]                          [.] 0000000000000000
+    3.95%     0.00%  Chrome_InProcRe  [unknown]                          [.] 0x0000000042400000
+    3.21%     0.00%  Chrome_InProcRe  libwebviewchromium.so              [.] 0xffffffff2f83abe8
+    3.20%     3.20%  Chrome_InProcRe  libwebviewchromium.so              [.] 0x000000000048abe8
+    2.74%     0.00%  Chrome_InProcRe  [unknown]                          [.] 0x0000000000000006
+    2.66%     0.00%  Chrome_InProcRe  libwebviewchromium.so              [.] 0xffffffff2f834afc
+    2.65%     2.65%  Chrome_InProcRe  libwebviewchromium.so              [.] 0x0000000000484afc
+    2.58%     0.00%  Chrome_InProcRe  [unknown]                          [.] 0x00000000ffcd2edc
+    2.37%     0.00%  Chrome_InProcRe  libwebviewchromium.so              [.] 0xffffffff2f83ac3a
+    2.36%     2.36%  Chrome_InProcRe  libwebviewchromium.so              [.] 0x000000000048ac3a
+    1.99%     0.00%  Chrome_InProcRe  libwebviewchromium.so              [.] 0xffffffff2f81453a
+    1.99%     1.99%  Chrome_InProcRe  libwebviewchromium.so              [.] 0x000000000046453a
+    1.99%     0.08%  mali-cmar-backe  [kernel.kallsyms]                  [k] el0_svc_naked
+    1.89%     0.00%  owser.barebones  [unknown]                          [k] 0000000000000000
+    1.57%     0.00%  Chrome_InProcGp  [unknown]                          [.] 0000000000000000
+    1.44%     0.00%  Thread-59        [unknown]                          [.] 0000000000000000
+    1.41%     1.41%  Chrome_InProcRe  libwebviewchromium.so              [.] 0x000000000048ac44
+    1.41%     0.00%  Chrome_InProcRe  libwebviewchromium.so              [.] 0xffffffff2f83ac44
+    1.25%     0.00%  mali-cmar-backe  libc.so                            [.] 0xffffffff195bc67c
+    1.23%     0.01%  mali-cmar-backe  [kernel.kallsyms]                  [k] compat_sys_ioctl
+    1.17%     0.03%  mali-cmar-backe  [kernel.kallsyms]                  [k] kbase_ioctl
+    1.14%     1.14%  Chrome_InProcRe  libwebviewchromium.so              [.] 0x000000000048abaa
+    1.14%     0.00%  Chrome_InProcRe  libwebviewchromium.so              [.] 0xffffffff2f83abaa
+    1.09%     0.00%  Chrome_InProcRe  [unknown]                          [.] 0x0000000056cd985d
+    1.08%     0.03%  mali-cmar-backe  [kernel.kallsyms]                  [k] kbase_jd_submit
+    1.04%     0.00%  Chrome_InProcRe  libwebviewchromium.so              [.] 0xffffffff2f836356
+    1.04%     1.04%  Chrome_InProcRe  libwebviewchromium.so              [.] 0x0000000000486356
+    1.04%     0.00%  Chrome_InProcRe  libwebviewchromium.so              [.] 0xffffffff2f834ae8
+    1.03%     1.03%  Chrome_InProcRe  libwebviewchromium.so              [.] 0x0000000000484ae8
+    1.01%     0.03%  owser.barebones  [kernel.kallsyms]                  [k] el0_svc_naked
+    0.99%     0.00%  Chrome_InProcRe  libc.so                            [.] 0xffffffff1958b4b8
```

上图有‘+’的地方可以用‘enter’键来遍历其调用关系。

3.7 FlameGraph

还可以通过一些脚本来方便分析调用关系，Flame Graph就是一个比较好用的可视化分析工具。

下载：

```
$ git clone https://github.com/brendangregg/FlameGraph.git
```

生成图形：

```
perf script --vmlinux=<kernel_folder>/vmlinux --symfs $ANDROID_PRODUCT_OUT/symbols -i
perf.data | FlameGraph/stackcollapse-perf.pl | FlameGraph/flamegraph.pl > flamegraph.html
```

4 在Linux平台使用

arm版本的linux发行版很多都没有提供Perf的包，所以需要自己手动编译一个Perf，由于Perf依赖的elfutils/binutils/zlib，所以实际上需要交叉编译四个东西。

首先编译zlib，源码地址

```
$ CC=aarch64-linux-gnu-gcc ./configure --
prefix=/home/cmc/workspace/linaro/toolchain/armlinux/aarch64/gcc-linaro-6.3.1-2017.02-
x86_64_aarch64-linux-gnu/aarch64-linux-gnu/libc/usr
$ make && make install
```

Note: prefix要指向你的交叉编译工具的库目录

编译elfutils，我直接用的最新的版本的：

```
$ git clone git://sourceware.org/git/elfutils.git
```

配置：

```
$ cd /path/to/elfutils
$ mkdir build
$ ./configure --enable-maintainer-mode --host=aarch64-linux-gnu --
prefix=/home/cmc/workspace/linaro/elfutils/build
```

修改Makefile： 删除elfutils根目录下Makefile里面的libcpu

修改backends/Makefile： 删除backends/Makefile中的libebl_i386和libebl_x86_64有关的所有东西

编译：

```
$ make && make install
```

编译binutils，这个要考虑和gcc版本的兼容，我用的2.28.1的版本，源代码地址

```
$ cd /path/to/binutils
$ mkdir build
$ ../configure --target=aarch64-linux-gnu --host=aarch64-linux-gnu --
prefix=/home/cmc/workspace/linaro/binutils-2.28.1/build
$ make && make install
```

编译Perf，Perf是Kernel一起发布的，所以直接下载一个Kernel就有了，但是交叉编译的话，需要改一些东西：

修改Makefile.perf，在前面加入：

```
EXTRA_CFLAGS=-I/path/to/elfutils/build/inclue -L/path/to/elfutils/build/lib -
I/path/to/binutils/build/include -L/path/to/binutils/build/lib
WERROR=0
NO_LIBPERL=1
NO_LIBPYTHON=1
```

编译

```
$ cd /path/to/kernel/tools/perf
$ make -f Makefile.perf perf ARCH=arm64
CROSS_COMPILE=/home/cmc/workspace/linaro/toolchain/armlinux/aarch64/gcc-linaro-6.3.1-2017.02-
x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu- -j8
```

理论上在arm的linux发行版上直接编译Perf应该也是可以的，但是我没有试过。用法的话和Android是一样的，这里就不叙说了。

5 Simpleperf使用

Android 7.0开始提供了一个更完整的Perf版本Simpleperf：

```
$ source build/envsetup.sh
$ lunch
$ mmma system/extras/simpleperf
```

Simpleperf相对之前google移植的Perf有以下改进

- 支持剖析apk中兼容的共享库，从 .gnu_debugdata 段读取符号表和调试信息
- 提供更方便分析的脚本
- 纯静态，所以和Android版本无关，只要指令集兼容都能跑

ndk r13开始就提供了Simpleperf工具，所以也可以直接下载编译好的工具：

```
$ git clone https://aosp.tuna.tsinghua.edu.cn/platform/prebuilts/simpleperf
```

用法上和Perf是类似的，命令基本通用，可以直接参考上面Perf的命令。

Simpleperf更多信息， 特别是调试java程序的方法，请参考官方手册

1. 后面也会简单介绍一些Simpleperf↩

2. 这个是指top统计信息的刷新间隔而不是采样间隔↩