平头哥

T-HEAD CPU perf使用说明

2020·02·29

Perf 简介

平头哥

- Perf是一系列强大的性能分析工具集合。
- 在Linux 2.6.31版本引入，至今tool/perf目录拥有1万多个提交，是内核开发中最活跃的几个领域之一。
- Perf最初只负责处理系统性能事件，随着版本迭代和演进也引入了诸如probe，tracepotint，auxtrace，bpf等等各具特色的子工具。
- 通过perf可以使用一到两行命令就完成像程序热点采样，接口调用分析，阻塞分析这些以往需要插入大量分析代码才能完成的事情。

Perf 简介

平头哥

- 借助于内核日渐健全的tracepoint，perf拥有了一千多个linux内核预插桩点，可以对系统中调度，内存，文件系统，网络等各方面进行分析
- 围绕perf和系统性能事件，也有不少像perf-tool，火焰图，热点图，vtune等第三方功能扩展。
- 本次培训主要针对perf stat/record/report，硬件PMU，火焰图几个部分重点进行介绍

Perf trace流程

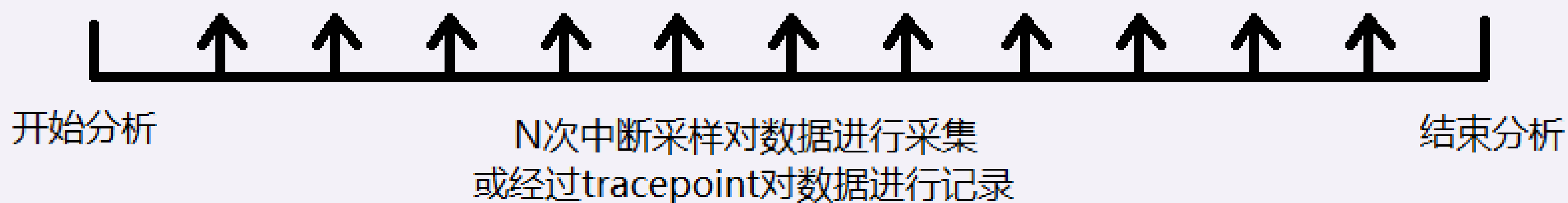
- Perf 进行性能分析的方式通常有两种：

  1. 使用perf stat等命令对特定的事件计数器进行计算，并在程序结束后打印数值

  2. 使用perf record等命令以若干的事件为触发间隔对系统进行采样，将数据保存至perf.data文件以供后续分析

```
# perf stat ls
messages

 Performance counter stats for 'ls':

             42.83 msec task-clock                #    0.735 CPUs utilized
                 0      context-switches          #    0.000 K/sec
                 0      cpu-migrations            #    0.000 K/sec
                59      page-faults               #    0.001 M/sec
           2563283      cycles                    #    0.060 GHz
            714777      instructions              #    0.28  insn per cycle
            109487      conditionl-branch-instructions #    2.556 M/sec
             13290      conditional-branch-misspredict #   12.14% of all branches

       0.058298533 seconds time elapsed

       0.012837000 seconds user
       0.051349000 seconds sys
```

```
# perf record ls
messages           perf.data        perf.data.old
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.011 MB perf.data (260 samples) ]
#
```



Perf 事件支持

平头哥

- 通过Perf list可以查看当前支持的所有事件包含硬件事件，软件事件，硬件cache事件，PMU事件以及预设Tracepoint事件

```
# perf list

List of pre-defined events (to be used in -e):

  L1-icache-access                                   [Hardware event]
  L1-icache-misses                                   [Hardware event]
  conditional-branch-instructions                    [Hardware event]
  conditional-branch-misspredict                     [Hardware event]
  cpu-cycles OR cycles                               [Hardware event]
  d-utlb-misses                                      [Hardware event]
  i-utlb-misses                                      [Hardware event]
  indirect-branch-instructions                       [Hardware event]
  indirect-branch-mispredict                         [Hardware event]
  instructions                                       [Hardware event]
  jtlb-misses                                        [Hardware event]
  lsu-cross-4k-stall-counter                         [Hardware event]
  lsu-other-stall-counter                            [Hardware event]
  lsu-speculation-fail                               [Hardware event]
  lsu-sq-data-discard-counter                        [Hardware event]
  lsu-sq-discard-counter                             [Hardware event]
  rf-instruction-counter                             [Hardware event]
  rf-launch-fail-counter                             [Hardware event]
  rf-reg-launch-fail-counter                         [Hardware event]
  store-instructions                                 [Hardware event]

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

  L1-dcache-load-misses                              [Hardware cache event]
  L1-dcache-loads                                    [Hardware cache event]
```

使用perf trace的准备工作

平头哥

- T-HEAD 平台支持通过buildroot来快速完成整个Linux系统的搭建，所以要在T-HEAD平台上体验perf相关功能需要通过 https://github.com/c-sky/buildroot/releases 获取最新的buildroot使用源代码编译或使用Quick Start中的命令下载预编译的镜像直接执行，具体可参照buildroot用户手册，perf功能在T-HEAD配置中默认开启，启动后可以输入perf命令确认环境

```
Starting network: OK
processor         : 0
C-SKY CPU model : ck810f
product info[0] : 0x0504000c
product info[1] : 0x10000000
product info[2] : 0x20000000
product info[3] : 0x30000000
hint (CPU funcs): 0x00000000
ccr  (L1C & MMU): 0x00000001
ccr2 (L2C)        : 0x00000000

arch-version : e68b1635ac2711d5fc939bce66318aa118c9484a

Skip the ci test

Welcome to Buildroot
buildroot login: root
# perf

 usage: perf [--version] [--help] [OPTIONS] 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 build-id cache.
```

使用火焰图分析函数热点

- 以callchain_test为例如，这里将介绍如何使用perf命令和火焰图分析一个程序的函数热点
  1）首先运行: perf record -g callchain_test 对callchain_test程序的热点和调用栈进行采样
  2）运行perf report可以直接观测函数热点，如果第一步中未使能-g选项则只显示热点不显示函数调用关系，可以看到右图中热点集中在test_4函数，函数调用入口为test_1
  3) 当函数热点较为分散时可以通过火焰图更直观的看到函数调用关系

```
perf report
To display the perf.data header info, please use --header/--header-only options.


Total Lost Samples: 0

Samples: 154  of event 'cpu-clock'
Event count (approx.): 38500000

Overhead  Command         Shared Object      Symbol
........  ..............  ................  ......................................

  55.19%  callchain_test  callchain_test     [.] test_4
   1.95%  callchain_test  [kernel.kallsyms]  [k] __softirqentry_text_start
   1.95%  callchain_test  [kernel.kallsyms]  [k] flush_tlb_one
   1.95%  callchain_test  [kernel.kallsyms]  [k] flush_tlb_range
   1.30%  callchain_test  [kernel.kallsyms]  [k] flush_tlb_mm
   1.30%  callchain_test  [kernel.kallsyms]  [k] free_hot_cold_page
   1.30%  callchain_test  [kernel.kallsyms]  [k] page_add_file_rmap
   1.30%  callchain_test  [kernel.kallsyms]  [k] path_openat
   1.30%  callchain_test  [kernel.kallsyms]  [k] sys_mmap_pgoff
   1.30%  callchain_test  ld-2.28.9000.so    [.] $t
   1.30%  callchain_test  libc-2.28.9000.so  [.] __cxa_atexit
   1.30%  perf            [kernel.kallsyms]  [k] perf_event_exec
   0.65%  callchain_test  [kernel.kallsyms]  [k] __d_lookup_rcu
   0.65%  callchain_test  [kernel.kallsyms]  [k] __fdget_raw
   0.65%  callchain_test  [kernel.kallsyms]  [k] __fput
   0.65%  callchain_test  [kernel.kallsyms]  [k] __kunmap_atomic
```

```
perf report
To display the perf.data header info, please use --header/--header-only options.


Total Lost Samples: 0

Samples: 164  of event 'cpu-clock'
Event count (approx.): 41000000

Children      Self  Command         Shared Object      Symbol
........  ........  ..............  ................  ..................................

  62.80%     0.00%  callchain_test  libc-2.28.9000.so  [.] __libc_start_main
            |
            ---__libc_start_main
               |
               |--59.76%--main
               |          test_1
               |          test_2
               |          test_3
               |          test_4
               |          |
               |           --0.61%--ret_from_exception
               |                     schedule
               |                     __schedule
               |                     finish_task_switch
               |
               |--2.44%--0x3132c
               |          |
```

使用火焰图分析函数热点

平头哥

4）下载火焰图工具

5）在第一步后运行命令perf script > perf.samples 生成可读的采样文件

6）之后通过网络或文件系统导出生成的perf.samples

7）在主机侧运行火焰图工具脚本

./stackcollapse-perf.pl perf.samples > perf.fold

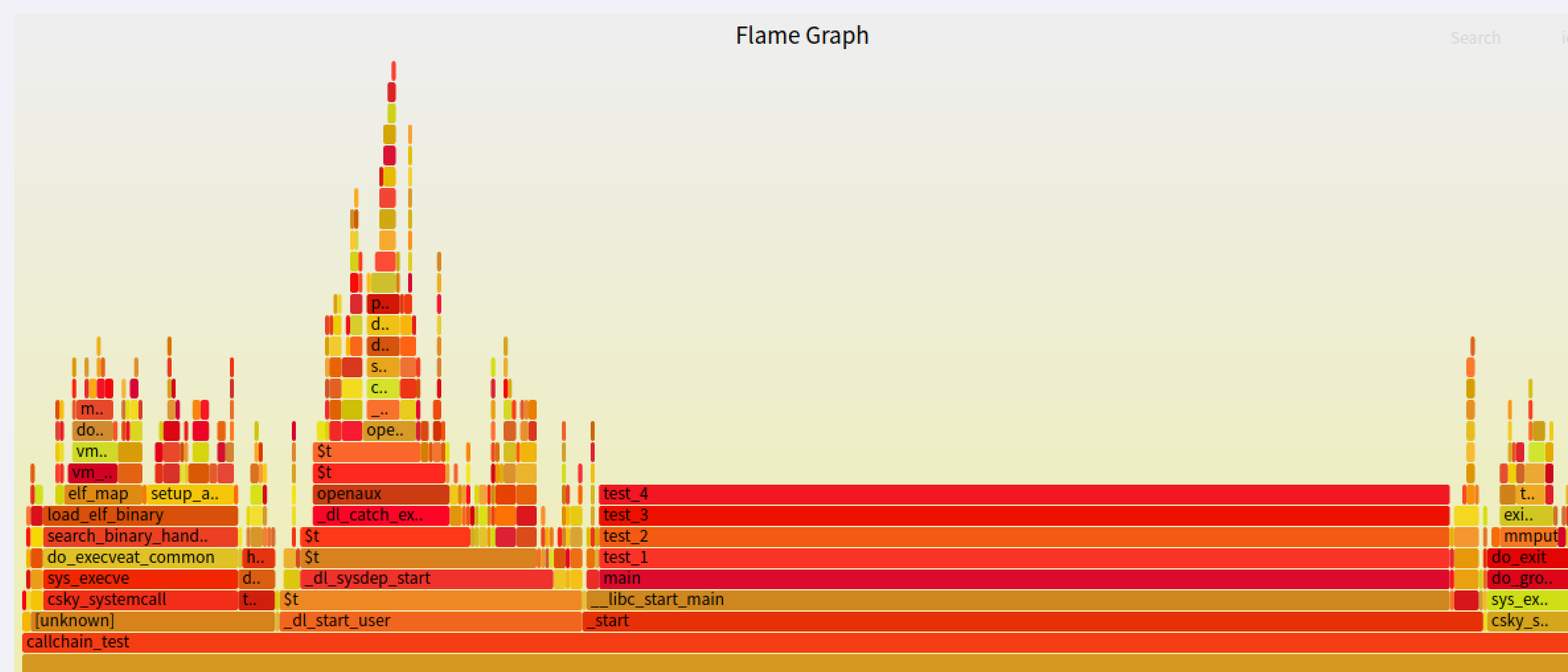
./flamegraph.pl perf.fold > perf.svg

最后即可通过浏览器打开火焰图

perf.svg

8）对照源文件即可发现代码test_4 中包含了大量无用的赋值运算可以优化

```c
#include <stdio.h>

void test_4(void)
{
  volatile int i, j;

  for(i = 0; i < 10000000; i++)
    j=i;
}

void test_3(void)
{
  volatile int i, j;
  test_4();
  for(i = 0; i < 3000; i++)
    j=i;
}
```



使用硬件PMU分析程序性能

平头哥

- T-HEAD CPU上支持了大量硬件事件计数器，包含了指令数，周期数，cache访问，分支预测等等，这里以memcpy为例演示硬件PMU的使用

  1）使用perf list可以列出所有支持的事件名称

  这里只使用instructions和cycles

  2）使用perf stat –e instructions, cycles tst-mem

  观查程序运行时的指令周期消耗

  3）增大TST_SIZE后重试perf stat，

  可以看到重复的memcpy执行

  会拥有更高的IPC,代码可以通过

  减少低IPC部分的占比来提升总体效率

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define TST_SIZE 64*1024

int main()
{
    char *bufa, *bufb;
    int i;

    bufa = malloc(TST_SIZE);
    bufb = malloc(TST_SIZE);

    for (i = 0; i < TST_SIZE; i++) {
        bufa[i] = i;
    }

    for (i = 0; i < 10; i++) {
        memcpy(bufb, bufa, TST_SIZE);
    }

    free(bufa);
    free(bufb);
}
```

```
# perf stat -e instructions,cycles ./tst-mem

 Performance counter stats for './tst-mem':

           1784503      instructions              #    0.72  insn per cycle
           2471128      cycles

       0.101622134 seconds time elapsed

       0.058743000 seconds user
       0.054826000 seconds sys

# perf stat -e instructions,cycles ./tst-mem2

 Performance counter stats for './tst-mem2':

         328971329      instructions              #    1.06  insn per cycle
         309475863      cycles

      10.344352812 seconds time elapsed

       9.293407000 seconds user
       1.051254000 seconds sys
```

使用硬件PMU分析程序性能

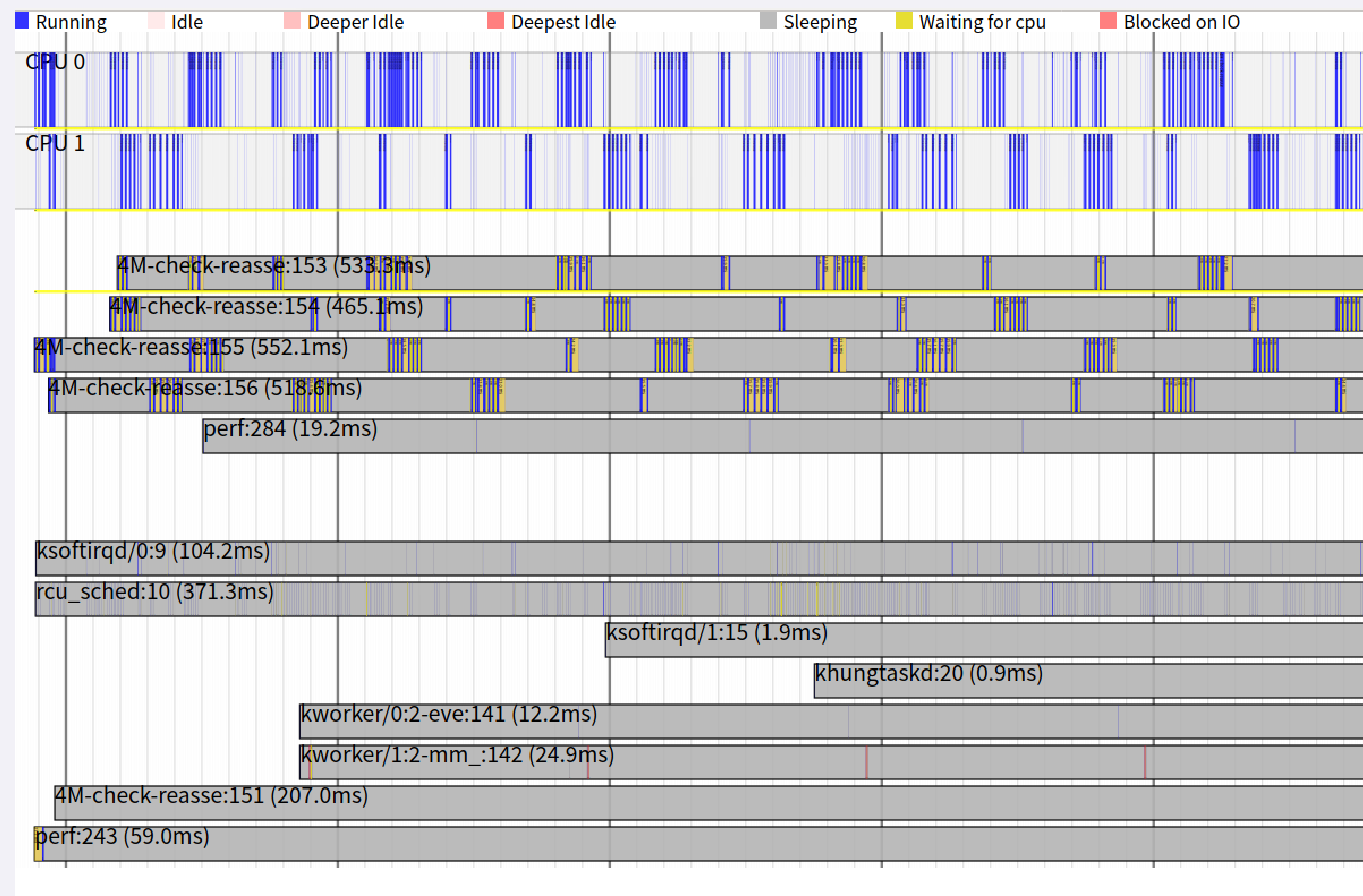
平头哥

- 硬件PMU信息不但可以通过perf stat进行输出，也可以通过perf record采样具体硬件事件的触发点
- 如使用命令perf record –e L1-icache-misses 就可以分析指令cache miss主要在哪些地方出现 进而分析_etext函数的指令为什么会被踢出cache，如何避免这些cache miss函数的访问

```
# perf record -e L1-icache-misses ./tst-mem
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.011 MB perf.data (256 samples) ]
# perf report
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 256  of event 'L1-icache-misses'
# Event count (approx.): 15493
#
# Overhead  Command  Shared Object      Symbol
# ........  .......  .................  ......................................
#
    11.80%  tst-mem  [kernel.kallsyms]  [k] _etext
     7.21%  tst-mem  [kernel.kallsyms]  [k] _raw_spin_unlock_irqrestore
     2.08%  tst-mem  [kernel.kallsyms]  [k] perf_event_mmap
     2.05%  tst-mem  [kernel.kallsyms]  [k] filemap_map_pages
     1.81%  tst-mem  [kernel.kallsyms]  [k] kmem_cache_free
     1.75%  tst-mem  [kernel.kallsyms]  [k] ptep_clear_flush
     1.59%  tst-mem  [kernel.kallsyms]  [k] do_page_fault
     1.39%  tst-mem  [kernel.kallsyms]  [k] __mod_node_page_state
     1.39%  tst-mem  [kernel.kallsyms]  [k] mmap_region
     1.37%  tst-mem  [kernel.kallsyms]  [k] __handle_mm_fault
     1.28%  tst-mem  [kernel.kallsyms]  [k] prep_new_page
#
```

Timechart

- 生成Timechart通常分三步
  - a)后台执行测试程序：
  - b)执行：perf timechart record
  - c)执行：perf timechart 生成svg文件
- 可以对系统上程序运行和切换进行采样并生成运行时间图
- 适用于程序占用，进程切换以及CPU/IO停滞分析，例如对文件系统的访问为什么会特别久是不是卡在IO上还是程序内部进入了睡眠



Buildroot配置

- 需要使用perf功能首先可以通过Buildroot的Kernel>Linux Kernel Tools选单选中perf工具，可以勾选TUI来使用字符交互界面

```
Kernel > Linux Kernel Tools
                              Linux Kernel Tools
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus
  ----).  Highlighted letters are hotkeys.  Pressing <Y> selects a feature, while <N>
  excludes a feature.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend:
  [*] feature is selected  [ ] feature is excluded

        [ ] cpupower
        [ ] gpio
        [ ] iio
        [ ] pci
        -*- perf
        [ ]   enable perf TUI
        [ ] selftests
        [ ] tmon
```

Buildroot配置

- 如果需要函数符合显示和翻栈功能需要配置Target packages > Libraries > Other下的elfutils函数库支持

```
> Target packages > Libraries > Other
                              Other
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus
 ----).  Highlighted letters are hotkeys.  Pressing <Y> selects a feature, while <N
 excludes a feature.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Lege
 [*] feature is selected  [ ] feature is excluded
          ^(-)
          [ ] eigen
          [*] elfutils
          [ ]   Install programs (NEW)
```

函数库需要通过Build options选单下配置带调试信息编译，且不被strip target binaries

```
    Mirrors and Download locations  --->
(0) Number of jobs to run simultaneously (0 for auto)
[ ] Enable compiler cache
[*] build packages with debugging symbols
      gcc debug level (debug level 2)  --->
[ ] strip target binaries
    gcc optimization level (optimize for size)  --->
    libraries (both static and shared)  --->
```

Buildroot配置

- 如果需要使用函数调用栈分析功能时，需要添加额外编译选项，-mbacktrace编译用于支持FP回溯功能，而-fexceptions用于支持DWARF unwind回溯机制。9系列不需要添加参数。

```
    *** Toolchain Generic Options ***
[ ] Copy gconv libraries
(-mbacktrace -fexceptions) Target Optimizations
()  Target linker options
```

- 如果annotate功能给采样点添加反汇编上下文显示，则需要选择编译Target packages > Development tools 选单下的binutils来提供bfd和opcode库支持

```
Target packages > Development tools
                              Development tools
 Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
 Highlighted letters are hotkeys.  Pressing <Y> selects a feature, while <N> excludes a
 feature.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] feature
 is selected  [ ] feature is excluded

        [ ] bats
        [*] binutils
        [*]   binutils binaries
        [ ] bsdiff
```

Linux内核配置

平头哥

- 需要使用硬件PMU时，可以通过Linux内核的Device Drivers > Performance monitor support选单选择可以使用的PMU单元

```
Device Drivers > Performance monitor support
                         Performance monitor support
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submer
  ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> exclude
  modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

         [*] C910 Performance Monitoring Unit
```

- 需要使用一些系统内置tracepoint时(如系统调用)是，可以通过Kernel hacking > Tracers选单配置对应tracepoint，perf ftrace功能也依赖Kernel function tracers使能

```
Kernel hacking > Tracers
                              Tracers
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus
  ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M>
  modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

        --- Tracers
        [*]   Kernel Function Tracer
        [*]     Kernel Function Graph Tracer
        [ ]   Enable trace events for preempt and irq disable/enable
        [ ]   Interrupts-off Latency Tracer
        [ ]   Scheduling Latency Tracer
        [ ]   Tracer to detect hardware latencies (like SMIs)
        [*]   Trace syscalls
        [ ]   Create a snapshot trace buffer
```

Linux内核配置

平头哥

- 需要perf probe插桩功能时，可以通过Linux内核的General architecture-dependent options选单选择kprobe功能

```
> General architecture-dependent options
            General architecture-dependent options
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
  submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
  includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
  exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]

      [*] Kprobes
      [*] Stack Protector buffer overflow detection
      [*]   Strong Stack Protector
```

- 并通过/搜索确认UPROBES，UPROBE_EVENTS功能已经被默认打开

```
Symbol: UPROBES [=y]
Type  : bool
Defined at arch/Kconfig:120
  Depends on: ARCH_SUPPORTS_UPROBES [=y]
Selected by [y]:
  - UPROBE_EVENTS [=y] && TRACING_SUPPORT [=y] && FTRACE [=y] && ARCH_S


Symbol: UPROBE_EVENTS [=y]
```

配置更改后的重新编译

- 当在buildroot环境下对内核、perf工具、文件系统进行修改之后，可以通过以下方式进行重新编译以保证各个部分按照预期进行了更新

  1）当内核配置通过make linux-menuconfig进行修改之后，可以直接在编译目录下make编译内核；对内核代码进行修改后则需要删除build/linux-*.*.*/.stamp_built后再执行make编译内核

  2）当perf工具依赖的elfutils，binutils工具发生变化时，需要删除build/linux-tools目录并进入build/linux-*.*.*/tools/perf对perf工具环境进行clean，以保证perf工具被完整重新编译

  3）当要为需要调试的程序添加调试信息时，需要按照13页对buildroot选项进行配置，并手动删除build/“软件包”目录，使得目标软件包重新编译并安装至文件系统

Perf 常用功能

平头哥

- annotate 解析perf.data并生成采样点周围的反汇编，标注采样点和跳转指令
- diff 用于比较两个perf.data在采样点占比区别
- list 列出所有支持perf事件
- record 执行一个命令通过中断收集相关profile信息
- report 解析perf.data并生成采样点分布报告
- stat 执行一个命令收集它的性能计数器统计信息
- top 显示当前系统运行热点及对应函数
- timechart 生成一段时间内系统运行的图形化输出
- script 将perf.data转换为可读采样数据

全局火焰图记录

平头哥

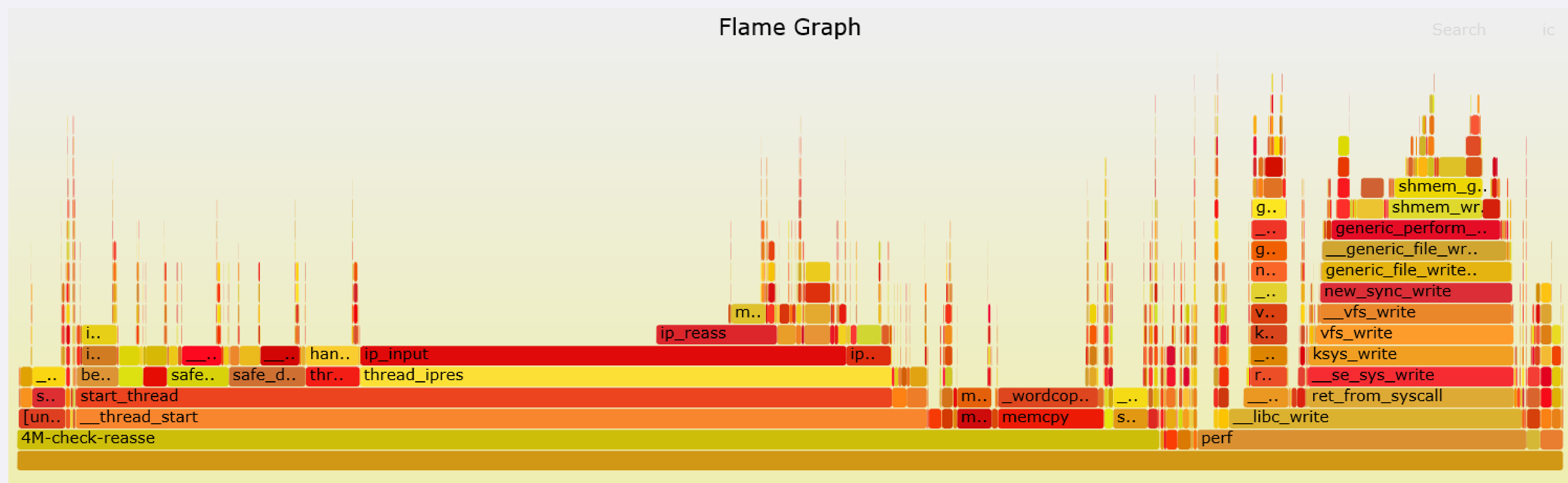
1 后台启动测试程序：./test_case

2 运行perf record -g 记录全局信息

3 看到程序执行结束后ctrl+c掐停perf

4 之后按照第8页第四点之后流程生成火焰图即可对所有程序进行追踪

5 如果要监控某个cpu可以添加参数—cpu <cpu>



Perf ftrace

- Perf ftrace提供了一种较为便利的对程序运行时函数调用关系和时间进行追踪的功能
- 支持function和function graph两种模式，默认的为function graph模式
- 可以通过参数配置目标CPU, 需要过滤的函数调用，PID等等参数更精确的来分析目标程序

```
# perf ftrace ls
 0)                 |  check_and_switch_context() {
 0) ! 123.880 us    |    _raw_spin_lock_irqsave();
 0)   4.240 us      |    _raw_spin_unlock_irqrestore();
 0)   3.220 us      |    _raw_spin_lock_irqsave();
 0)   3.040 us      |    _raw_spin_unlock_irqrestore();
 0)   3.020 us      |    _raw_spin_lock_irqsave();
 0)   3.020 us      |    _raw_spin_unlock_irqrestore();
 0)   3.060 us      |    _raw_spin_lock_irqsave();
 0)   3.020 us      |    _raw_spin_unlock_irqrestore();
 0) # 1054.020 us   |  }
 0) ! 724.020 us    |  flush_icache_deferred();
 ------------------------------------------
 0)   <...>-127    =>    perf-128
 ------------------------------------------

 0)   5.780 us      |  finish_task_switch();
 0)                 |  prepare_to_wait_event() {
 0)   3.640 us      |    _raw_spin_lock_irqsave();
```

```
perf ftrace -t function -N _raw_spin_lock_irqsave ls
         <...>-137   [000] d...   177.171512: check_and_switch_context <-__schedule
         <...>-137   [000] d...   177.172098: _raw_spin_unlock_irqrestore <-atomic64_read
         <...>-137   [000] d...   177.172153: _raw_spin_unlock_irqrestore <-atomic64_read
         <...>-137   [000] d...   177.172158: _raw_spin_unlock_irqrestore <-atomic64_read
         <...>-137   [000] d...   177.172163: _raw_spin_unlock_irqrestore <-atomic64_cmpxch

         <...>-137   [000] d...   177.172426: flush_icache_deferred <-__schedule
          perf-138   [000] d...   177.172992: finish_task_switch <-__schedule
          perf-138   [000] d...   177.173021: csky_do_IRQ <-csky_irq
          perf-138   [000] d...   177.173037: csky_mpintc_handler <-csky_do_IRQ
          perf-138   [000] d...   177.173069: __handle_domain_irq <-csky_mpintc_handler
          perf-138   [000] d...   177.173096: irq_enter <-__handle_domain_irq
          perf-138   [000] d...   177.173113: rcu_irq_enter <-irq_enter
```

Perf ftrace选项

- -a, --all-cpus 全CPU的数据追踪，不需要添加目标程序
- -C, --cpu <cpu> 配置CPU Mask来对跟踪的核进行过滤
- -D, --graph-depth <n> 设置function graph最大的跟踪深度，避免显示过长超出屏幕
- -G, --graph-funcs <func> 设置graph过滤，不显示选择函数以外的函数
- -g, --nograph-funcs <func>设置nograph过滤， 过滤只显示选择函数以外的函数
- -N, --notrace-funcs <func> 不追踪给定的函数调用
- -p, --pid <pid> 按照一个现有的进程id进行跟踪，附加到一个正在执行的程序
- -T, --trace-funcs <func> 只追踪给定的函数调用
- -t, --tracer <tracer> 配置追踪模式为 function_graph或function
- -v, --verbose 显示一些执行过程的调试信息

Perf probe

平头哥

- Perf probe提供了一种动态向内核或应用程序中插入桩点，观测函数调用的参数，返回值的功能
- Perf probe –a可以通过各类参数指定添加事件的名称，函数名，偏移，参数，代码行数等等各类属性
- perf probe插入的桩点需要通过perf record命令如：perf record -e probe:sys_openat ls进行数据采集
- Perf probe –l可以列出现有的所以probe事件
- Perf probe –d可以用于移除对应的事件

```
#  perf probe -a "sys_openat dfd=%a0 filename=%a1"
Added new event:
  probe:sys_openat     (on sys_openat with dfd=%a0 filename=%a1)

You can now use it in all perf tools, such as:

        perf record -e probe:sys_openat -aR sleep 1

# perf record -e probe:sys_openat ls
Couldn't synthesize bpf events.
perf.data      perf.data.old
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.002 MB perf.data (5 samples) ]
# perf report --stdio
# To display the perf.data header info, please use --header/--header-onl
#
#
# Total Lost Samples: 0
#
# Samples: 5  of event 'probe:sys_openat'
# Event count (approx.): 5
#
# Overhead  Trace output
# ........  ........................................
#
    40.00%  (8019f3ec) dfd=0xffffff9c filename=0x7fdfb218
    20.00%  (8019f3ec) dfd=0xffffff9c filename=0x2aac5118
    20.00%  (8019f3ec) dfd=0xffffff9c filename=0x7fdfb20c
    20.00%  (8019f3ec) dfd=0xffffff9c filename=0xd1a9d
```

Perf probe

平头哥

- 通过指定内核镜像和源代码路径的方式，可以支持查看特定函数的参数，向特定的C代码行插入桩点的功能
- Perf probe –L命令可以通过函数名快速的找到对应的代码和相关参数
- 当向应用程序插入桩点时需要通过-x指定对应的镜像文件如：perf probe -x /lib/libc.so.6 memcpy
- Perf probe –D可以显示对应probe点的详细信息

```
# perf probe -F *arch_cpu_idle*
arch_cpu_idle
arch_cpu_idle_dead
arch_cpu_idle_enter
arch_cpu_idle_exit
arch_cpu_idle_prepare
#  perf probe -L arch_cpu_idle -k /mnt/tst/vmlinux -s /mnt/tst/linux-custom/
<arch_cpu_idle@/mnt/tst/linux-custom//arch/csky/kernel/process.c:0>
      0  void arch_cpu_idle(void)
      1  {
         #ifdef CONFIG_CPU_PM_WAIT
                asm volatile("wait\n");
         #endif

         #ifdef CONFIG_CPU_PM_DOZE
                asm volatile("doze\n");
         #endif

         #ifdef CONFIG_CPU_PM_STOP
                asm volatile("stop\n");
         #endif
                local_irq_enable();
     14  }
         #endif
```

```
#  perf probe -V sys_openat -k /mnt/tst/vmlinux -s /mnt/tst/linux-custom/
Available variables at sys_openat
        @<sys_openat+0>
                long int        dfd
                long int        filename
                long int        flags
                long int        mode
# perf probe -x /lib/libc.so.6 memcpy
Added new event:
  probe_libc:memcpy    (on memcpy in /lib/libc-2.28.9000.so)

You can now use it in all perf tools, such as:

        perf record -e probe_libc:memcpy -aR sleep 1
```

参考资料

平头哥

- https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/tools/perf/Documentation?h=v5.5.11
- http://www.brendangregg.com/perf.html