
C-SKY CPF 用户手册

Release v1.3

2018 年 11 月 26 日

Copyright © 2018 杭州中天微系统有限公司，保留所有权利。

本文件的产权属于杭州中天微系统有限公司（下称中天公司）。本文件仅能分布给：(i) 拥有合法雇佣关系，并需要本文件的信息的中天微系统员工，或 (ii) 非中天微组织但拥有合法合作关系，并且其需要本文件的信息的合作方。对于本文件，禁止任何在专利、版权或商业秘密过程中，授予或暗示的可以使用该文件。在没有得到杭州中天微系统有限公司的书面许可前，不得复制本文件的任何部分，传播、转录、储存在检索系统中或翻译成任何语言或计算机语言。

商标申明

杭州中天微系统的 LOGO 和其它所有商标归杭州中天微系统有限公司所有，所有其它产品或服务名称归其所有者拥有。

注意

您购买的产品、服务或特性等应受中天公司商业合同和条款的约束，本文件中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，中天公司对本文件内容不做任何明示或默示的声明或保证。由于产品版本升级或其他原因，本文件内容会不定期进行更新。除非另有约定，本文件仅作为使用指导，本文件中的所有陈述、信息和建议不构成任何明示或暗示的担保。

Copyright © 2018 Hangzhou C-SKY MicroSystems Co.,Ltd. All rights reserved.

This document is the property of Hangzhou C-SKY MicroSystems Co.,Ltd. This document may only be distributed to: (i) a C-SKY party having a legitimate business need for the information contained herein, or (ii) a non-C-SKY party having a legitimate business need for the information contained herein. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document. No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise without the prior written permission of C-SKY MicroSystems Co.,Ltd.

Trademarks and Permissions

The C-SKY Logo and all other trademarks indicated as such herein are trademarks of Hangzhou C-SKY MicroSystems Co.,Ltd. All other products or service names are the property of their respective owners.

Notice

The purchased products, services and features are stipulated by the contract made between C-SKY and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied. The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

杭州中天微系统有限公司 C-SKY MicroSystems Co.,LTD

地址: 杭州市西湖区西斗门路 3 号天堂软件园 A 座 15 楼

邮编: 310012

网址: www.c-sky.com

Contents:

| | |
|---|-----------|
| 第一章 C-SKY Profiling analysis utils (CPF) | 1 |
| 1.1 简介 | 1 |
| 1.2 工具介绍 | 1 |
| 1.3 框架结构 | 2 |
| 第二章 编译 | 3 |
| 2.1 推荐环境 | 3 |
| 2.2 获取源码或可执行程序 | 3 |
| 2.3 从源码编译 | 3 |
| 第三章 快速上手 | 4 |
| 3.1 简易示例 | 4 |
| 第四章 trace: 数据生成 | 6 |
| 4.1 trace 文件内容 | 6 |
| 4.2 trace 生成方法 | 6 |
| 第五章 record: trace 记录 | 8 |
| 5.1 record 选项介绍 | 8 |
| 5.2 record 使用方法 | 8 |
| 第六章 top: 实时分析 | 9 |
| 6.1 top 选项介绍 | 9 |
| 6.2 top 使用方法 | 10 |
| 第七章 report: 离线分析 | 12 |
| 7.1 report 选项介绍 | 12 |
| 7.2 report 使用方法 | 12 |
| 第八章 stat: 统计分析 | 15 |
| 8.1 stat 选项介绍 | 15 |
| 8.2 stat 使用方法 | 15 |
| 第九章 cabinet: 缓存解析 | 18 |
| 9.1 cabinet 选项介绍 | 18 |
| 9.2 cabinet 使用方法 | 19 |
| 第十章 objdump: 反汇编工具 | 21 |

| | |
|-----------------------------|-----------|
| 10.1 objdump 选项介绍 | 21 |
| 10.2 objdump 使用方法 | 21 |
| 第十一章 config: 配置与查看配置 | 23 |
| 11.1 配置与查看配置 | 23 |
| 11.2 可配置选项 | 23 |
| 11.3 默认配置 | 24 |
| 11.4 配置示例 | 24 |
| 第十二章 附录 | 25 |
| 12.1 示例代码源码 | 25 |
| 索引 | 27 |

第一章 C-SKY Profiling analysis utils (CPF)

1.1 简介

CPF(C-SKY Profiling analysis utils) 是一套针对 csky cpu 应用程序的性能分析与调试调优工具，包含了一系列分析工具。CPF 基于 csky trace 进行分析，非侵入式的获取 cpu 运行时的信息，真实反映 cpu 的运行情况。

CPF 的功能与特性:

1. 支持非侵入式的获取硬件或者模拟器产生的 trace。
2. 支持实时和离线两种方式分析 trace 文件。
3. 多种输出方式，支持 TUI、stdio 和 json 文件等多种显示和输出格式。
4. 可分析函数热点、cache 命中、分支预测、指令统计等 cpu 运行情况。
5. 可灵活配置 cache 等 cpu 配置。
6. 可配合调试器使用。

1.2 工具介绍

CPF 以工具集的方式整合来多个用于性能分析和调试调优的工具，包括但不限于：

表 1.1: CPF 工具清单

| 工具名 | 基本命令 | 功能 |
|---------|-------------------------------|-----------------------------------|
| top | cpf top -e foo.elf | 动态显示程序运行中的函数级 profiling 信息。 |
| record | cpf record -e foo.elf | 接收来自硬件的 trace 流，压缩并保存到本地。 |
| report | cpf report -e foo.elf | 离线分析 trace 信息，生成函数级 profiling 信息。 |
| stat | cpf stat -e foo.elf | 离线分析 trace 信息，生成各项静态指标的统计信息。 |
| objdump | cpf objdump -p -i foo.elf | 根据起始和结束地址反汇编目标 elf 文件。 |
| cabinet | cpf cabinet -s 0 -l 1024 -n 2 | 根据起始地址和长度，读取对应块的 profiling 缓存。 |
| config | cpf config model.cpu=ck803 | 查看和配置 cpf 的.cpfconfig 文件。 |
| version | cpf version | 查看 cpf 的版本信息。 |
| help | cpf help | 查看 cpf 的帮助信息。 |

1.3 框架结构

CPF 是一套彼此之间松耦合的工具集，不同的工具针对性能分析和调试调优的不同应用场景。在工具集之上，是用于显示和命令采集的交互层，CPF 支持命令行/TUI/IDE 3 种交互方式。工具集之下，是用于分析和统计的分析层，分析层包括 trace 接收/trace 解析/时钟精确模型/反汇编库等不同的模块。

CPF 结构框架如下：

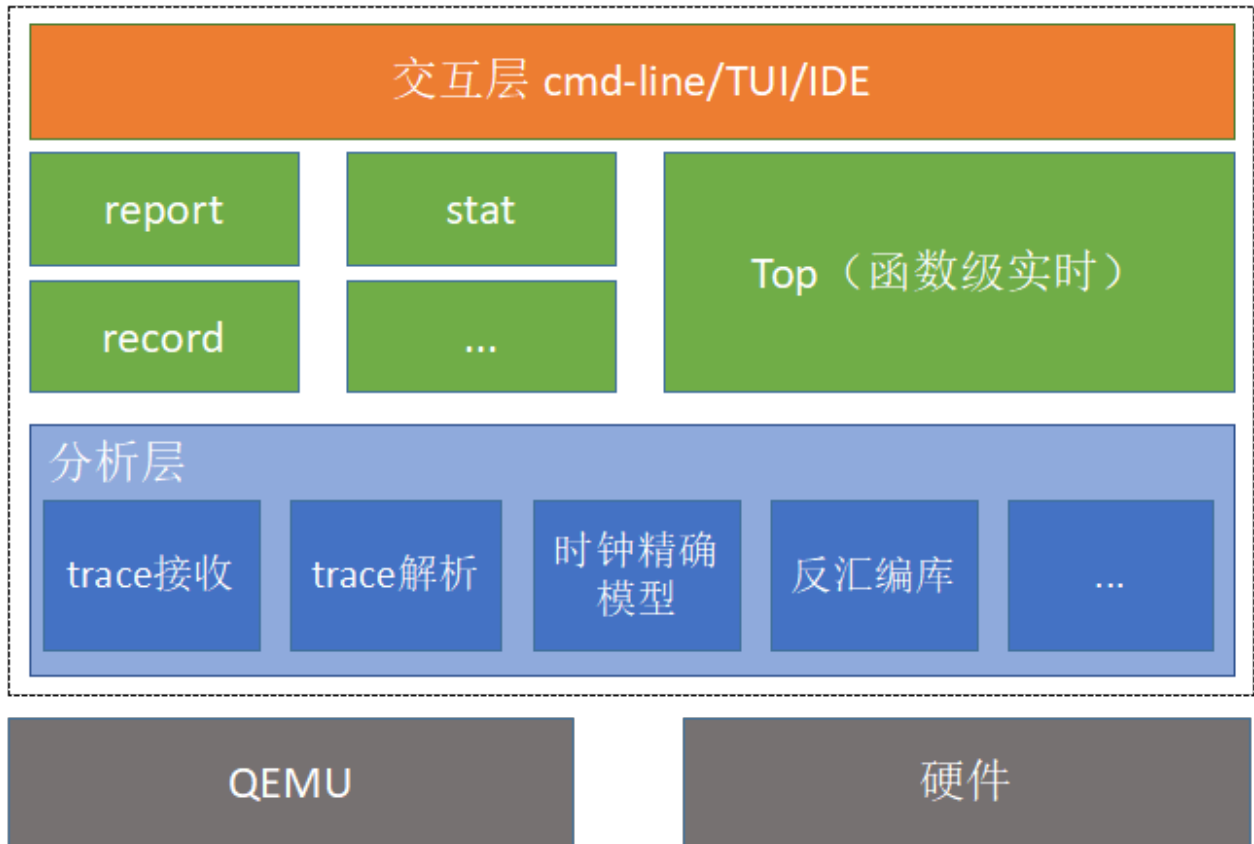


图 1.1: CPF 软件框架

第二章 编译

2.1 推荐环境

推荐使用以下操作系统版本：

- ubuntu 16.04 64 位

2.2 获取源码或可执行程序

CPF 可执行程序有以下几种获取方式：

1. 目前没有开放官方渠道，如有需要请邮件联系。

CPF 源码有以下几种获取方式：

1. 目前没有开放官方渠道，如有需要请邮件联系。

2.3 从源码编译

这节描述了如何从源码编译 CPF。

编译：

```
make
```

如果需要安装可执行程序到本机执行目录，则可以

安装：

```
make install DESTDIR={install dir}
```

第三章 快速上手

3.1 简易示例

下面以 Qemu 系统模式环境运行 `cpf/test/test_elf/system_mode/ck803efr1/` 目录下的 `hello world` 程序，来演示 CPF record 和 report 工具的使用。

step 1:

Qemu 系统模式运行 `hello_world.elf` 应用程序，并在命令行中添加 `-CPF` 和 `-S` 选项。此时 Qemu 进入等待连接状态。

```
qemu-system-cskyv2 -machine smartl -kernel hello_world.elf -nographic -CPF -S
```

更多 Qemu 支持的 CPF 相关选项，请查看 `qemu help`。

step 2:

CPF 运行 record 工具，记录 trace 信息。

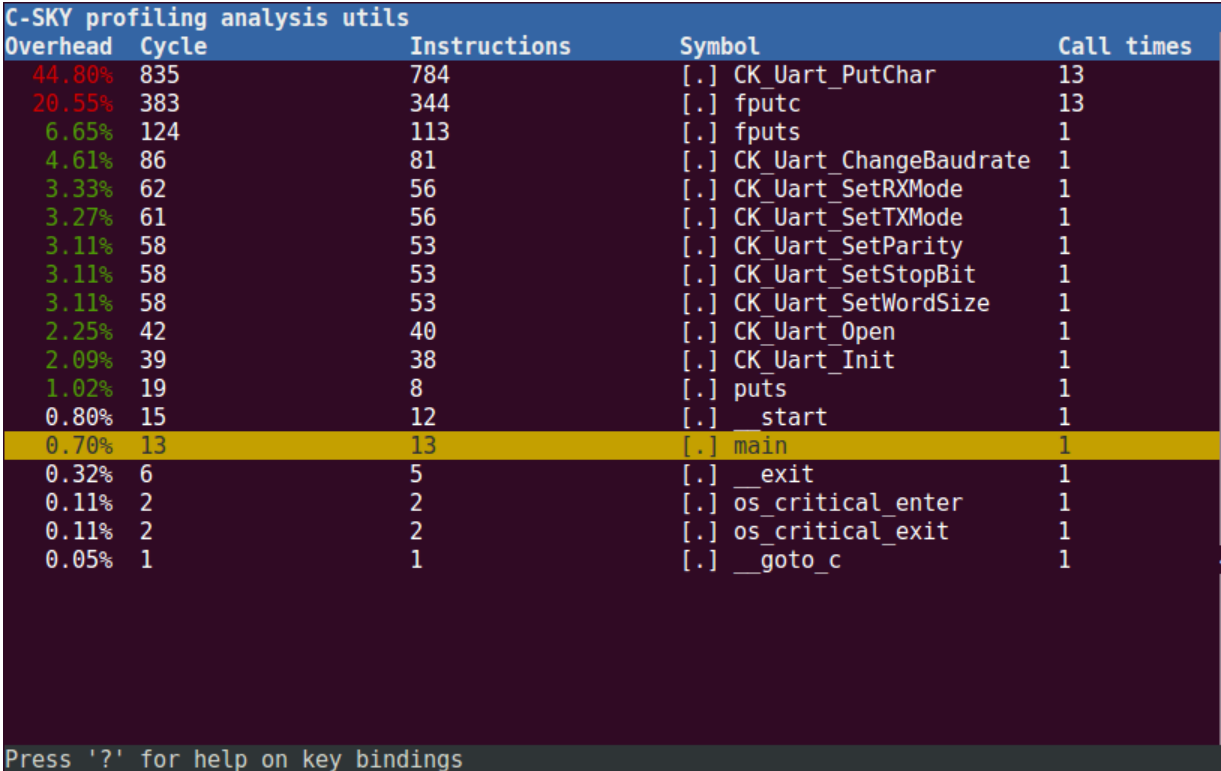
```
cpf record -e hello_world.elf
```

step 3:

CPF 运行 report 工具，进行 trace 数据的分析。

```
cpf report -e hello_world.elf
```

运行 report 后，会在 TUI 界面显示函数级统计信息，显示如下：



```
C-SKY profiling analysis utils
Overhead  Cycle      Instructions      Symbol              Call times
44.80%   835        784               [.] CK_Uart_PutChar 13
20.55%   383        344               [.] fputc            13
6.65%    124        113               [.] fputs            1
4.61%    86         81                [.] CK_Uart_ChangeBaudrate 1
3.33%    62         56                [.] CK_Uart_SetRXMode 1
3.27%    61         56                [.] CK_Uart_SetTXMode 1
3.11%    58         53                [.] CK_Uart_SetParity 1
3.11%    58         53                [.] CK_Uart_SetStopBit 1
3.11%    58         53                [.] CK_Uart_SetWordSize 1
2.25%    42         40                [.] CK_Uart_Open      1
2.09%    39         38                [.] CK_Uart_Init      1
1.02%    19         8                 [.] puts              1
0.80%    15         12                [.] __start           1
0.70%    13         13                [.] main              1
0.32%    6          5                 [.] __exit            1
0.11%    2          2                 [.] os_critical_enter 1
0.11%    2          2                 [.] os_critical_exit  1
0.05%    1          1                 [.] __goto_c          1

Press '?' for help on key bindings
```

| Overhead | Cycle | Instructions | Symbol | Call times |
|----------|-------|--------------|----------------------------|------------|
| 44.80% | 835 | 784 | [.] CK_Uart_PutChar | 13 |
| 20.55% | 383 | 344 | [.] fputc | 13 |
| 6.65% | 124 | 113 | [.] fputs | 1 |
| 4.61% | 86 | 81 | [.] CK_Uart_ChangeBaudrate | 1 |
| 3.33% | 62 | 56 | [.] CK_Uart_SetRXMode | 1 |
| 3.27% | 61 | 56 | [.] CK_Uart_SetTXMode | 1 |
| 3.11% | 58 | 53 | [.] CK_Uart_SetParity | 1 |
| 3.11% | 58 | 53 | [.] CK_Uart_SetStopBit | 1 |
| 3.11% | 58 | 53 | [.] CK_Uart_SetWordSize | 1 |
| 2.25% | 42 | 40 | [.] CK_Uart_Open | 1 |
| 2.09% | 39 | 38 | [.] CK_Uart_Init | 1 |
| 1.02% | 19 | 8 | [.] puts | 1 |
| 0.80% | 15 | 12 | [.] __start | 1 |
| 0.70% | 13 | 13 | [.] main | 1 |
| 0.32% | 6 | 5 | [.] __exit | 1 |
| 0.11% | 2 | 2 | [.] os_critical_enter | 1 |
| 0.11% | 2 | 2 | [.] os_critical_exit | 1 |
| 0.05% | 1 | 1 | [.] __goto_c | 1 |

图 3.1: hello world 程序的 report 输出

第四章 trace: 数据生成

cpu 运行的过程中会产生各种各样的行为，在 trace 协议中，某些特定的 cpu 行为被定义为事件，例如分支起始事件、数据存储事件、中断异常等，这些事件为 cpf 工具的分析提供了必要信息。

trace 信息通过硬件或者模拟器的 trace 模块产生，并通过 socket 方式发送。cpf 工具通过底层的 record 模块接收 trace 信息，用于后续的分析。

4.1 trace 文件内容

记录的二进制文件由两部分组成，一部分是大小为 4k 的头部 (Header)，另一部分是 trace 信息。

Header 的内容包括：

- Record 版本
- trace 版本
- Host 机信息
- 可执行文件路径
- 记录日期时间
- 记录选项
- CPU 信息
- Board 信息
- 事件信息
- Header 结束标志

4.2 trace 生成方法

trace 流由 qemu 生成，通常生成的命令如下：

trace 生成参数 -csky-trace

用户模式

```
qemu-cskyv2 -csky-trace port=8810,tb_trace=on,mem_trace=off,auto_trace=on hello_world.elf
```

注：用户模式中，elf 文件必须是 qemu 参数的最后一个。

系统模式

```
qemu-system-cskyv2 -machine smartl -kernel hello_world.elf -nographic -csky-trace port=8810,tb_
↪trace=on,mem_trace=off,auto_trace=on -S
```

简单参数 -CPF

‘-CPF’ 选项是 ‘-csky-trace port=8810,tb_trace=on,mem_trace=off,auto_trace=on’ 选项的缩写，使得 qemu 的使用更加方便。

用户模式

```
qemu-cskyv2 -CPF hello_world.elf
```

系统模式

```
qemu-system-cskyv2 -machine smartl -kernel hello_world.elf -nographic -CPF -S
```

选项含义

表 4.1: qemu trace 产生选项

| 选项 | 说明 |
|------------|---------------------|
| port | trace 使用的 TCP 端口 |
| tb_trace | 最基本 trace 的能力，默认 on |
| mem_trace | 数据 trace 的能力，默认 off |
| auto_trace | 设置使能位，默认 on |

使用 trace 控制寄存器

trace 模块支持使用控制寄存器实现精确 trace。在使用控制寄存器精确控制 trace 时，auto_trace 选项应该设置为 off。

```
1 /* 开启指令 */
2 __asm__(
3     "movi a0, 3 \n\t      \
4     mtrcr a0, cr<1, 14> \n\t \
5     movi a0, 1 \n\t      \
6     mtrcr a0, cr<0, 14>\n\t \
7     ");
8 /* 关闭指令 */
9 __asm__(
10    "movi a0, 0 \n\t      \
11    mtrcr a0, cr<0, 14> \n\t \
12    ");
```

第五章 record: trace 记录

CPF record 工具通过 socket 接口接收硬件或者模拟器 (Qemu) 产生的 trace 数据。record 工具简单分析 trace 数据，并把 trace 数据压缩后通过文件形式保存在本地（默认文件名：cpfddata/cpf.data），方便后续分析。

5.1 record 选项介绍

以下是一些 record 工具常用选项，更多选项可以参考 `cpf record -h`。

`-h/--help`

显示 record 工具的帮助信息。

`-e/--elf <file>`

选择 trace 数据对应的程序。

`-o/--output <file>`

选择 trace 数据保存的路径和文件名，默认文件名 `cpfddata/cpf.data`。

`-p/--port <n>`

设置服务端的 socket 端口，默认 8810。

`-s/--server <ip>`

设置服务端的 ip 地址，默认 127.0.0.1。

5.2 record 使用方法

cpf record 工具需要配合硬件或者模拟器使用，需要先启动具有 trace 模块的硬件或者启动模拟器，进入等待连接状态。

然后启动 `cpf record` 建立连接，连接建立后程序将会启动，record 工具会把 trace 数据压缩并保存在本地：

step 1:

```
qemu-system-cskyv2 -machine smartl -kernel hello_world.elf -nographic -CPF -S
```

step 2:

```
cpf record -e hello_world.elf -s 127.0.0.1 -p 8810 -o cpfddata/cpf.data
```

第六章 top: 实时分析

cpf top 工具通过 socket 与硬件或者模拟器连接，能够实时分析并显示执行过程中函数执行情况，包括函数名、调用次数、指令数、指令周期数等，以类似 linux top 命令的方式显示。top 工具支持按照指令数或指令周期数排序，方便用于分析程序运行过程中的热点函数。

cpf top 工具默认 1 秒的频率进行刷新，显示过去 1 秒内函数执行的情况，最小刷新周期为 100ms，最小精度为 100ms。目前 top 不具备 TUI 显示功能，可以直接向终端输出或者通过 json 文件输出。

6.1 top 选项介绍

以下是一些 record 工具常用选项，更多选项可以参考 `cpf top -h`。

-h/--help

显示 top 工具的帮助信息。

-e/--elf <file>

选择 trace 数据对应的程序。

-g/--callgraph

选择 top 工具是否显示函数间的调用关系。

-m/--ms <n>

设置 top 工具的刷新频率，最小周期 100ms，最小精度 100ms，默认 1000ms。

-p/--port <n>

设置服务端的 socket 端口，默认 8810。

-s/--server <ip>

设置服务端的 ip 地址，默认 127.0.0.1。

--dir <directory>

设置 top 工具产生文件的保存目录，默认 cpfddata。

--json

top 产生数据使用 json 格式输出。

--disable-dump

不进行反汇编文件输出。

--disable-save-prof

不进行缓存文件的输出，该缓存文件提供给 cabinet 工具使用，用于 profiling 的快照分析。

```
--stream <file>
```

选择 top 输出信息的位置，默认为 STDOUT。

6.2 top 使用方法

下面以 cpf/test/trace/ 目录下的 foo.c 为例，演示 cpf top 工具的使用方法和解释显示结果的含义。

6.2.1 操作步骤：

step 1: 编译目标程序 foo.elf，源码请参考该目录下的 foo.c 文件。

```
make
```

step 2: 模拟器用户模式运行 foo.elf 程序，需要添加 -CPF 选项。

```
qemu-cskyv2 -CPF foo.elf
```

step 3: CPF 运行 top 工具，实时分析 trace 数据，并显示。

```
列表格式: cpf top -e foo.elf -g
```

```
json 格式: cpf top -e foo.elf -g --json --stream foo.json
```

6.2.2 操作结果：

通过 cpf top 工具，可以得到 foo.elf 程序在 0-1 秒和 1-2 秒内的函数执行情况（由于篇幅限制，只截取来一部分），结果如下图：

```
# cpf top: foo.elf
# Update: 1000ms Tot Cycles: 27497299 Tot insts: 25048027
# Read Bytes: 18151706 Write Bytes: 12258091

# Overhead      Cycles      Insts      Calls      Symbol
# .....      .....      .....      .....      .....
#
 67.15%    18465444    16056908    401423    longa
 20.70%    5692337    5689661     2677     foo1
 10.48%    2882052    2879376     2676     foo2
  1.51%    417703     385567     5356     memset
  0.13%    37480      34802         1     main
  0.01%     662        466         1     __uClibc_main
  0.01%    250        220         8     memcpy
  0.01%    249        208         2     __libc_setup_tls
```

(continues on next page)

(续上页)

```
[...]  
  
# cpf top: foo.elf  
# Update: 2000ms Tot Cycles: 54938384 Tot insts: 50045217  
# Read Bytes: 18114215 Write Bytes: 12232837  
  
# Overhead      Cycles      Insts      Calls      Symbol  
# .....      .....      .....      .....      .....  
#  
  67.15%    18429362    16025532    400638    longa  
  20.70%    5680986     5678315     2671     foo1  
  10.48%    2876667     2873996     2671     foo2  
   1.51%    416676      384624      5342     memset  
   0.13%    37394       34723        0       main  
  
[...]
```

每次输出包含 title 和函数列表 2 个部分。

title 部分包含来应用程序名、时间戳、总的指令执行情况和内存访问情况。

函数列表分为 5 列，显示来这个时间周期内的函数运行情况：

- 第一列是函数运行周期的占比；
- 第二列是函数运行周期；
- 第三列是函数指令数；
- 第四列是函数调用次数；
- 第五列是函数名。

除了在终端打印的函数列表，在 cpfddata/ 文件夹下还会生成如下 5 个文件：

- callgraph.json: json 格式组织的函数调用关系文件；
- cpf.data: 压缩保存的 trace 数据；
- cpf.prof: 分析结果的快照文件，可以通过 `-disable-save-prof` 选项不产生该文件；
- functions.json: json 格式组织的函数 profiling 信息；
- obj.dump: elf 文件的反汇编文件，可以通过 `-disable-dump` 选项不产生该文件；
- overview.json: json 格式组织的统计概要文件，显示了运行期间总的 profiling 信息。

第七章 report: 离线分析

CPF report 工具离线分析本地的 trace 数据文件，生成 trace 数据的函数级统计信息，可以方便地分析程序运行过程中的热点代码。

7.1 report 选项介绍

以下是一些 report 工具常用选项，更多选项可以参考 `cpf report -h`。

`-h/--help`

显示 report 工具的帮助信息。

`-e/--elf <file>`

选择需要分析的程序。

`-i/--input <file>`

选择 trace 文件的路径，默认 `cpfdata/cpf.data`。

`-g/--callgraph`

选择 top 工具是否显示函数间的调用关系。

`-L/--percent-limit <percent>`

只显示时钟周期占比大于 percent 的函数。

`--stdio`

以 stdio 接口显示 profiling 结果。

`--tui`

以 tui 接口显示 profiling 结果。

7.2 report 使用方法

cpf report 工具以 `cpf top` 或 `cpf record` 产生的 trace 文件为基础，通过 trace 解析、反汇编源程序、cpu 模型分析等获得程序的 profiling 信息，并以函数为单位进行显示。cpf report 的使用方法如下：

```
cpf report -e hello_world.elf -g
```

cpf report 运行 `cpf/test/test_elf/system_mode/ck803efr1/` 目录下的 hello world 程序，以 stdio 方式显示的结果如下：


```
# cpf report: hello_world.elf
# Tot Cycles: 1864 Tot insts: 1714
# Read Bytes: 1543 Write Bytes: 781

# Entry 0
# Overhead      Cycles      Insts      Calls      Symbol      caller
# .....      .....      .....      .....      .....      .....
#
  1.18%         22          18         1      __start      NULL
  0.70%         13          13         1      main         __start
  1.02%         19           8         1      puts         main
  6.65%        124         113         1      fputs        puts
  0.11%         2            2         1      os_critical_enter  fputs
 18.94%        353         318        12      fputc        fputs
  2.25%         42          40         1      CK_Uart_Open  fputc
  2.09%         39          38         1      CK_Uart_Init  CK_Uart_Open
  4.61%         86          81         1      CK_Uart_ChangeBaudrate  CK_Uart_Init
  3.11%         58          53         1      CK_Uart_SetParity  CK_Uart_Init
  3.11%         58          53         1      CK_Uart_SetWordSize  CK_Uart_Init
  3.11%         58          53         1      CK_Uart_SetStopBit  CK_Uart_Init
  3.33%         62          56         1      CK_Uart_SetRXMode  CK_Uart_Init
  3.27%         61          56         1      CK_Uart_SetTXMode  CK_Uart_Init
 41.20%        768         720        12      CK_Uart_PutChar  fputc
  0.11%         2            2         1      os_critical_exit  fputs
  1.61%         30          26         1      fputc        puts
  3.59%         67          64         1      CK_Uart_PutChar  fputc
```

report 输出包含 title 和函数列表 2 个部分。

title 部分包含来应用程序名、总的指令执行情况和内存访问情况。

函数列表按照异常入口进行分类，例如上面例子中的“Entry 0”表示显示的函数均为 0 号异常（重启异常）下的函数，如果有其它中断异常将会有其它 Entry。函数列表包含 5 到 6 列，显示来这个时间周期内的函数运行情况：

- 第一列是函数运行周期的占比；
- 第二列是函数运行周期，TUI 模式下根据占比使用不同的颜色进行着色，默认超过 10% 使用红色，1%-10% 使用绿色，低于 1% 白色显示；
- 第三列是函数指令数；
- 第四列是函数调用次数；
- 第五列是函数名，TUI 模式下通过 “[]” 表示函数类型，“[.]” 表示用户态函数，“[k]” 表示内核态函数；
- 第六列是当前函数的父函数，NULL 表示程序入口，无父函数，该列当添加 -g 选项时出现。

在 TUI 模式下，可以通过“?” 快捷键显示更多功能，例如通过“i” 快捷键显示 trace 文件的基本信息：

```

C-SKY profiling analysis utils
Overhead  Cycle      Instructions      Symbol      Call times
44.80%  835      Help              13
20.55%  383      h/?/F1           Show this window 13
6.65%   124      UP/DOWN/PGUP     1
4.61%   86       PGDN/SPACE       Navigate          1
3.33%   62       q/ESC/CTRL+C     Exit browser      1
3.27%   61
3.11%   58       For symbolic views (--sort has sym): 1
3.11%   58
3.11%   58      ENTER           Zoom into DS0/Threads & Annotate current symbol 1
2.25%   42      ESC             Zoom out          1
2.09%   39      a              Annotate current symbol 1
1.02%   19      d              Zoom into current DS0 1
0.80%   15      H              Display column headers 1
0.70%   13      L              Change percent limit 1
0.32%   6        m              Display context menu 1
0.11%   2        P              Print histograms to cpf.hist.N 1
0.11%   2        i              Show header information 1
0.05%   1        /              Filter symbol by name 1

Press any key...

Press '?' for help on key bindings

```

图 7.1: report help 菜单

```

Header information
# hostname: ubuntu
# os release: 4.4.0-127-generic
# arch: x86_64
# cpf version: 1.0
# cpu (actually used): ck803
# cpu(from cpf record --cpu): (null)
# cpu(from trace cpuid): ck803
# elf filename: /home/chenw/Desktop/cpf/test/test_elf/system_mode/ck803efr1/hello_world.elf
# record time: Tue Aug 28 09:50:29 2018
# record from: 127.0.0.1:8810
# trace version: 1
# trace id: 248
# trace cpuid: 0x04900003
# trace filename: /home/chenw/Desktop/cpf/test/test_elf/system_mode/ck803efr1/cpfdata/cpf
# record cmdline: cpf record -e hello_world.elf

Press 'q' to exit

```

图 7.2: report info 选项

第八章 stat: 统计分析

cpf stat 的主要功能为统计 trace 二进制文件中的 trace 事件, 主要包括执行的周期数、指令数、分支命中情况、cache 命中情况、异常情况和各类指令数等。

8.1 stat 选项介绍

以下是一些 stat 工具常用选项, 更多选项可以参考 `cpf stat -h`。

`-h/--help`

显示 stat 工具的帮助信息。

`-e/--elf <file>`

选择需要分析的程序。

`-i/--input <file>`

选择 trace 文件的路径, 默认 `cpfddata/cpf.data`。

`--seq <file>`

统计 file 中指定的指令序列。

8.2 stat 使用方法

cpf stat 工具以 `cpf top` 或 `cpf record` 产生的 trace 文件为基础, 通过 trace 解析、反汇编源程序、cpu 模型分析等获得程序的 profiling 信息, 并显示各项统计指标的 profiling 信息。cpf stat 运行 `cpf/test/test_elf/system_mode/ck803efr1/` 目录下的 `hello world` 程序, 并统计统计 `addi` 指令 (两个寄存器都为 `sp`) 之后为 `rts` 指令的指令序列的执行数量, stat 的命令如下:

```
cpf stat -e hello_world.elf --seq seq.txt
```

cpf stat 执行结果如下:

```
# cpf stat: hello_world.elf
#
# .....
```

```
total cycles:      1864
```

(continues on next page)

(续上页)

```
total instructions: 1714
CPI: 1.088
Read bytes: 1543
Write bytes: 781

[ icache ]
icache refs: 0
icache miss: 0
icache miss rate: 0.00%

[ dcache ]
dcache refs: 0 (0 rd + 0 wr)
dcache miss: 0 (0 rd + 0 wr)
dcache miss rate: 0.00% (0.00% + 0.00% )

[ branch ]
predicts: 120
predict miss: 52
predict miss rate:43.33%

[ events ]
exceptions: 0

[ instruction summary ]
load instructions: 414
store instructions: 200
abs_jump instructions: 62
branch instructions: 120
dsp instructions: 0
vdsp instructions: 0
float instructions: 0
rts instructions: 39
rte instructions: 0

[ instruction sequence match ]
instruction sequence:
.*addi.*sp,.*sp.*
.*rts.*

sequence matches: 35
```

stat 输出包含 8 部分内容，自上而下分别是：

1. stat 分析的应用程序名；
2. 指令数、周期数和内存访问情况；

3. icache 访问情况，包括 icache 访问次数和 icache miss 次数；
4. dcache 访问情况，包括 dcache 访问次数和 dcache miss 次数；
5. 分支预测情况，包括分支预测的次数和分支预测错误次数；
6. 事件统计情况，包括异常中断统计；
7. 各类指令统计情况；
8. 如果在 stat 中加入”-seq <file>”选项，可以统计指定指令序列的执行次数，其中文件需按照正则表达式书写。

第九章 cabinet: 缓存解析

cpf cabinet 工具用于解析 cpf top 工具产生的快照文件 cpf.prof, 快速获取指定时间范围的 profiling 信息, 为 IDE 使用提供便利。

cabinet 工具包括 2 个功能:

1. 输出从 start time 开始的 number 个 length 长度的时间片段的指令统计信息;
2. 输出从 start time 开始的 400 条 pctrace 信息。

9.1 cabinet 选项介绍

以下是一些 cabinet 工具常用选项, 更多选项可以参考 `cpf cabinet -h`。

-h/--help

显示 objdump 工具的帮助信息。

-i/--input <file>

选择快照文件, 默认 `cpfddata/cpf.prof`。

-l/--length <n>

时间片段的长度, 最小单位和精度均为 1024。

-n/--num <n>

分析时间片段的数量。

-s/--start <n>

分析时间片段的起始时间。

--pctrace

输出 pc trace 信息, 此时不需要 `-l` 和 `-n` 选项。

--stream <file>

将输出重定位到指定文件, 默认 `STDOUT`。

--dir <directory>

选择快照文件所在文件夹, 默认 `cpfddata/.`

9.2 cabinet 使用方法

cpf cabinet 工具包括 2 个功能，分别是获取指令统计信息和 pc trace 信息，使用方法如下：

1. 获取指令统计信息

```
cpf cabinet -s 0 -l 1024 -n 2
```

以下是 0 时刻开始，2 个时钟周期为 1024 的时间片段的指令统计结果：

```
{"profiling information":  
  [  
    {"load":85,  
     "store":183,  
     "abs_jump":16,  
     "branch":145,  
     "dsp":0,  
     "vdsp":0,  
     "float":0,  
     "rts":5,  
     "rte":0  
    },  
    {"load":225,  
     "store":104,  
     "abs_jump":31,  
     "branch":73,  
     "dsp":0,  
     "vdsp":0,  
     "float":0,  
     "rts":20,  
     "rte":0  
    }  
  ]  
}
```

2. 获取 pctrace 信息

```
cpf cabinet -s 0 --pctrace
```

以下是 0 时刻开始的 400 条 pctrace 信息：

```
{"pc trace":  
  [  
    {"cycle":0,"pc":33020},  
    {"cycle":2,"pc":33026},  
    {"cycle":4,"pc":33032},
```

(continues on next page)

(续上页)

```
    {"cycle":13,"pc":33048},  
  
    [...]  
  
    {"cycle":2866,"pc":35376}  
  ]  
}
```


第十章 objdump: 反汇编工具

cpf objdump 工具用于应用程序的反汇编，原理上和 GNU Binutils 的 objdump 相同，但是 cpf objdump 工具增加了针对代码片段的 objdump 功能，能精确的帮助使用者分析和调试应用程序。

10.1 objdump 选项介绍

以下是一些 objdump 工具常用选项，更多选项可以参考 `cpf objdump -h`。

- h/--help**
显示 objdump 工具的帮助信息。
- i/--input <file>**
选择 objdump 的文件。
- s/--start <addr>**
选择 objdump 的开始地址，默认为 0x0。
- e/--end <addr>**
选择 objdump 的结束地址，默认为 0xFFFFFFFF。
- p/--print-objdump**
类似 objdump -S 的方式打印结果。
- a/--all**
dump 目标程序的所有段，包括数据段。

10.2 objdump 使用方法

cpf objdump 工具可以精确的 dump 指定范围内的指令，以 `cp pf/test/test_elf/system_mode/ck803efr1/` 目录下的 hello world 程序为例，dump 0x8b90 到 0x8ba2 范围的指令。

```
cpf objdump -i hello_world.elf -s 0x8b90 -e 0x8ba2 -p
```

objdump 的结果如下：

```
hello_world.elf:      file format elf32-csky-little
```

(continues on next page)

(续上页)

```
Disassembly of section .text:
```

```
00008b90 <.text+0xb90>:
```

```
#include "stdio.h"
```

```
int main()
```

```
{
```

```
8b90: 1422      subi     sp, sp, 8
```

```
8b92: dd0e2000 st.w    r8, (sp, 0)
```

```
8b96: ddee2001 st.w    r15, (sp, 0x4)
```

```
8b9a: 6e3b     mov     r8, sp
```

```
printf("hello world!\n");
```

```
8b9c: 1006     lrw    r0, 0x8c30 // 0x8bb4
```

```
8b9e: e000000d bsr    0x8bb8 // 0x8bb8
```

第十一章 config: 配置与查看配置

cpf config 是一个类似 git config 的功能，可以用来设置配置值和查看配置。

cpf config 命令会在会在 cpf 所在目录生成一个. 开头的隐藏文件.cpfconfig。

11.1 配置与查看配置

用户可以通过 2 种方式设置配置值，一种是直接打开并编辑.cpfconfig 文件，另一种是通过 cpf config 命令设置。

通过 cpf config 命令配置的方式如下：

```
cpf config [section.name[=value] ...]
```

同理，用户可以通过打开.cpfconfig 来查看配置，也可以通过 config 命令来查看配置：

```
cpf config -l
```

11.2 可配置选项

cpf config 支持的选项有：

```
[model]
  cpu = [ck801|ck802|ck803|ck803efr1|...] # cpu 型号设置

[memory]
  delay = [0...n] # 内存访问或 cache miss 导致的延时
  cache = [on|off|yes|on] # 是否有可配置 cache
  icache = [on|off|yes|on] # 是否有 icache
  dcache = [on|off|yes|on] # 是否有 dcache

[cache1]
  is_config = [Yes|NO] # cache1 配置是否可用
  level = [0|1|2] # 几级 cache
  type = [icache|dcache] # cache 类型
  size = [128|512|1024|2048|4096|...] # cache 大小，单位为字节
  ways = [1|2|4|8|...] # 多少组组相连
  block = [16|32|64|...] # cache block 大小
```

(continues on next page)

(续上页)

```

writeback = [writeback|writethrough] # cache 的写回方式
enable = [Yes|NO] # cache 是否使能
region0_addr = (0-0xffffffff) # 可高缓空间起始地址
region0_size = (0-0xffffffff) # 可高缓空间大小
region0_enable = [Yes|NO] # 可高缓空间使能
[cache2]
[...]
```

11.3 默认配置

下表描述了不同 cpu 的默认内存配置情况:

表 11.1: 内存默认配置

| CPU | icache | dcache | delay |
|-------|--------|--------|-------|
| ck801 | No | No | 0 |
| ck802 | No | No | 0 |
| ck803 | No | No | 0 |
| ck805 | No | No | 0 |
| ck807 | Yes | Yes | 0 |
| ck810 | Yes | Yes | 0 |

如果配置 cache, 则 cache 的默认配置为:

表 11.2: cache 默认配置

| type | 使能 | 级数 | size | 组相联 | block 大小 | 写回/写直 | 可高缓空间 |
|--------|-----|----|------|-----|----------|-------|----------------|
| icache | Yes | 1 | 4096 | 4 | 16 | 写回 | 0x0-0xffffffff |
| dcache | Yes | 1 | 4096 | 4 | 16 | 写回 | 0x0-0xffffffff |

11.4 配置示例

例如设置当前分析的 cpu 为 ck803:

```

cpf config model.cpu=ck803
```

查看配置:

```

cpf config -l

>> model.cpu=ck803
```

第十二章 附录

12.1 示例代码源码

hello_world.c 示例:

```
#include "stdio.h"

int main()
{
    printf("hello world!\n");
    return 0;
}
```

foo.c 示例:

```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"

char *a =
↪ "abcdefghijklmnopqrstuvwxyz0123456789abcdefghijklmnopqrstuvwxyz0123456789abcdefghijklmnopqrstuvwxyz
↪ ";

int longa(int i)
{
    int j = 2;

    while(j--);

    return a[i];
}

void foo2()
{
    int i, j;
```

(continues on next page)

(续上页)

```
    char b[101] = { 0 };
    for (i = 0 ; i < 50; i++) {
        b[i] = longa(i);
    }
}

void foo1()
{
    int i, j;
    char b[101] = { 0 };
    for (i = 0; i < 100; i++) {
        b[i] = longa(i);
    }
}

int main(void)
{
    int i;
    for (i = 0; i < 10000; i++) {
        foo1();
        foo2();
    }
}
```

索引

Symbols

-dir <directory>
 命令行选项, 9, 18

-disable-dump
 命令行选项, 9

-disable-save-prof
 命令行选项, 9

-json
 命令行选项, 9

-pctrace
 命令行选项, 18

-seq <file>
 命令行选项, 15

-stdio
 命令行选项, 12

-stream <file>
 命令行选项, 9, 18

-tui
 命令行选项, 12

-L/-percent-limit <percent>
 命令行选项, 12

-a/-all
 命令行选项, 21

-e/-elf <file>
 命令行选项, 8, 9, 12, 15

-e/-end <addr>
 命令行选项, 21

-g/-callgraph
 命令行选项, 9, 12

-h/-help
 命令行选项, 8, 9, 12, 15, 18, 21

-i/-input <file>
 命令行选项, 12, 15, 18, 21

-l/-length <n>
 命令行选项, 18

-m/-ms <n>
 命令行选项, 9

-n/-num <n>
 命令行选项, 18

-o/-output <file>
 命令行选项, 8

-p/-port <n>
 命令行选项, 8, 9

-p/-print-objdump
 命令行选项, 21

-s/-server <ip>
 命令行选项, 8, 9

-s/-start <addr>
 命令行选项, 21

-s/-start <n>
 命令行选项, 18

命令行选项

-dir <directory>, 9, 18

-disable-dump, 9

-disable-save-prof, 9

-json, 9

-pctrace, 18

-seq <file>, 15

-stdio, 12

-stream <file>, 9, 18

-tui, 12

-L/-percent-limit <percent>, 12

-a/-all, 21

-e/-elf <file>, 8, 9, 12, 15

-e/-end <addr>, 21

-g/-callgraph, 9, 12

-h/-help, 8, 9, 12, 15, 18, 21

-i/-input <file>, 12, 15, 18, 21

-l/-length <n>, 18

-m/-ms <n>, 9

-n/-num <n>, 18

-o/-output <file>, 8

-p/-port <n>, 8, 9

-p/--print-objdump, 21
-s/--server <ip>, 8, 9
-s/--start <addr>, 21
-s/--start <n>, 18