Qualcomm Technologies, Inc.

Qualcomm® Hexagon™ gprof Profiler

User Guide

80-N2040-29 Rev. E

November 21, 2018

Contents

1 Introduction

This document describes the Qualcomm® Hexagon™ gprof Profiler utility, which displays information on the execution history of a program written for the Hexagon processor. This document is a reference for experienced C programmers with assembly language experience.

1.1 Conventions

`Courier` font is used for computer text and code samples:

```
unsigned long long hexagon_sim_read_pcycles()
```

The following notation is used to define command syntax:

- Square brackets enclose optional items (e.g., `[`**`label`**`]`).
- **Bold** indicates literal symbols (e.g., **`[`***`comment`***`]`**).
- An ellipsis, `. . .`, follows items that can appear more than once.

1.2 Technical assistance

For assistance or clarification on information in this document, submit a case to Qualcomm Technologies, Inc. (QTI) at https://createpoint.qti.qualcomm.com/.

If you do not have access to the CDMATech Support website, register for access or send email to support.cdmatech@qti.qualcomm.com.

2 Overview

The gprof profiler includes the following features:

- It operates as a stand-alone text-based application, and it is compatible with the GNU gprof profiler.
- It displays function-level and call-graph profiling information, which shows the function execution times and caller/callee relationships in a profiled program.
- It works with any Hexagon target program (stand-alone, RTOS-based, single-threaded, or multi-threaded).

NOTE The profiler performs *post-mortem* processing of the target application: that is, it is used after the target application has completed executing.

2.1 Profile data files

The gprof profiler obtains its profiling information from data files that are generated by the Hexagon simulator. These files are called gmon files.

For more information on generating profile data files, see the *Qualcomm Hexagon Simulator User Guide*.

2.2 Profile reports

The profiling information generated by the gprof profiler is presented in the form of a report which presents the number of cycles executed in each function, the functions that each function called or was called by, and how many times each function was called.

Profile reports are generated in plain text format.

2.3 Feedback

If you have any comments or suggestions on how to improve the profiler (or this document), please send them to:

support.cdmatech@qti.qualcomm.com

2 Use the Profiler

The profiler is used after an application is executed on the simulator. The simulator generates one or more data files containing profiling information while it executes the target application. After the simulation is completed, the profiler inputs the generated profile data files and displays the profiling information.

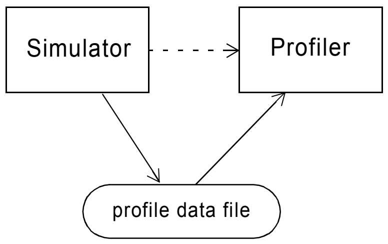

Figure 2-1 Using the profiler

The profiler operates as a stand-alone application. It displays two types of profile information:

- **Flat profile information** – The number of cycles executed in each function.
- **Call-graph information** – Which functions a given function called, which functions called a given function, and how many times each function was called.

All profiling information is displayed as text data, which can be redirected to an output file.

2.1 Create the profile data file

Before an application can be profiled, you must create a profile data file by simulating the application with the command option, `--profile`.

In some cases the profile data is distributed across multiple profile data files:

- When a program is a multi-threaded stand-alone program, its profile data files are named for each hardware thread (for example, `gmon.t_0`, `gmon.t_1`, ...).
- When a program is a multi-threaded RTOS application, its profile data files are named for each software thread (for example, `gmon.name1`, `gmon.name2`, ...).

In these cases, all of the profile data files must be specified on the profiler command line. To simplify this effort, the relevant command option accepts wild-card characters (for example, `gmon*`).

For more information on profile data files, see the *Hexagon Simulator User Guide*.

2.2 Start the profiler

To start the profiler from a command line, enter the following:

```
hexagon-gprof [executable_files...] [profile_data_files...]
                                                [> output_file]
```

2.2.1 Executable files

The profiler optionally accepts one or more executable program files as input files. The default is `a.out`.

The profiler uses the symbolic data in these files to create the profiling information.

NOTE When profiling an RTOS application system, each executable file in the system must be specified as an input file.

2.2.2 Profile data files

The profiler optionally accepts one or more profile data files as input files. The default is `gmon.out`.

All of the profile data files for the application must be specified. Multiple profile data files can be specified using wild-card characters (such as `gmon*`).

NOTE Processing multiple profile data files can be memory-intensive.

2.2.3 Output file

The profiler writes all the profiling information it generates to the standard output. The information can be optionally redirected to an output file.

2.3 Profile data

The profiler displays two types of profile information:

- **Flat profile information** – The number of cycles executed in each function.
- **Call-graph information** – Which functions a given function called, which functions called a given function, and how many times each function was called.

2.3.1 Flat profile

The *flat profile* lists the number of thread cycles executed in each function in a thread, with the list sorted by self cycles. For example:

```
Flat profile for t_3234404640_88:
  %     cumulative    self              self       total
 time    cycle(s)   cycle(s)  calls    Mc/call    Mc/call  name
  28.01        693        693      4       0.17       0.46  h264FrontEnd_CoreTaskHandler
  23.00       1262        569      5       0.11       0.13  VBUF_New
  14.23       1614        352      1       0.35       2.29  h264FrontEnd_ThreadProc
   4.20       1718        104      1       0.10       0.16  h264BatchDecoder_TryAcquireBatchServer
   3.88       1814         96     10       0.01       0.01  qurt_mutex_lock
   3.76       1907         93      4       0.02       0.03  VBUF_Release
   2.79       1976         69      5       0.01       0.02  qurt_anysignal_wait
   2.18       2030         54      4       0.01       0.01  h264_dec_query_input
   2.18       2084         54      1       0.05       0.06  qurt_anysignal_init
   2.18       2138         54      0                        qurt_trampoline
   2.18       2192         54      2       0.03       0.03  qurt_rmutex_lock
   2.18       2246         54      1       0.05       2.36  qurt_elite_thread_stub(void*)
   2.02       2296         50      1       0.05       0.05  __save_r16_through_r23
   1.78       2340         44     10       0.00       0.00  qurt_mutex_unlock
   1.09       2367         27      5       0.01       0.01  qurt_anysignal_clear
   0.93       2390         23      4       0.01       0.01  qurt_anysignal_wait_fatal
   0.65       2406         16      1       0.02       0.10  qurt_elite_queue_pop_front
   0.57       2420         14      1      0.01       0.02  __restore_r16_through_r23_and_deallocfr
   0.49       2432         12      8       0.00       0.00  qurt_anysignal_get
   0.49       2444         12      4       0.00       0.00  h264_dec_query_output
   0.32       2452          8      2       0.00       0.00  qurt_rmutex_unlock
   0.16       2456          4      1       0.00       0.00  __restore_r16_through_r17_and_deallocf
   0.12       2459          3      1       0.00       0.00  qurt_thread_set_ugp
   0.12       2462          3      1       0.00       0.00  qurt_futex_alloc_wait_queue
   0.12       2465          3      1       0.00       0.00  qurt_get_thread_id_in_tcb
   0.08       2467          2      1       0.00       0.00  qurt_thread_get_id
   0.04       2468          1      1       0.00       0.00  VBUF_GetInfo
```

Table 2-1 Values contained in the flat profile

Field	Description
`% time`	Percentage of the total number of cycles spent in a function
`cumulative cycle(s)`	Running total of cycles that includes the cycles for a specific function and all the functions above it in the list
`self cycle(s)`	Number of cycles spent in a function on the current thread
`calls`	Total number of times a function was called on the current thread
`self Mc/call`	Average number of cycles per function call
`total Mc/call`	Average number of self cycles plus descendant cycles per function call
`name`	Function name

2.3.2 Call graph

The *call graph* lists how many cycles were executed in each function plus its callers and callees, with the list sorted by self plus child cycles. For example:

```
Call graph for t_3234404640_88:

index   % time        self   children    called     name
-----------------------------------------------------------------
[1]      100.0          54       2420       0       qurt_trampoline [1]
                        54       2302      1/1      qurt_elite_thread_stub(void*) [1]
                        54          3      1/1      qurt_anysignal_init [2]
                         3          0      1/1      qurt_thread_set_ugp [3]
                         3          0      1/1      qurt_get_thread_id_in_tcb [4]
-----------------------------------------------------------------
                        54       2420      1/1      qurt_trampoline [1]
[2]       95.2          54       2302       1       qurt_elite_thread_stub(void*) [2]
                       352       1940      1/1      h264FrontEnd_ThreadProc [1]
                        54          0      1/2      qurt_rmutex_lock [2]
                         8          0      1/2      qurt_rmutex_unlock [3]
                         2          0      1/1      qurt_thread_get_id [4]
-----------------------------------------------------------------
                        54       2302      1/1      qurt_elite_thread_stub(void*) [1]
[3]       92.6         352       1940       1       h264FrontEnd_ThreadProc [3]
                       693       1141      4/4      h264FrontEnd_CoreTaskHandler [1]
                        69         23      5/5      qurt_anysignal_wait [2]
                        12          0      8/8      qurt_anysignal_get [3]
-----------------------------------------------------------------
                       352       1940      4/4      h264FrontEnd_ThreadProc [1]
[4]       74.1         693       1141       4       h264FrontEnd_CoreTaskHandler [4]
                       569        100      5/5      VBUF_New [1]
                       104         55      1/1      h264BatchDecoder_TryAcquireBatchServer [2]
                        93         32      4/4      VBUF_Release [3]
                        16         82      1/1      qurt_elite_queue_pop_front [4]
                        54          0      4/4      h264_dec_query_input [5]
                        27          0      4/5      qurt_anysignal_clear [6]
                        12          0      4/4      h264_dec_query_output [7]
                         1          0      1/1      VBUF_GetInfo [8]
-----------------------------------------------------------------
                       693       1141      5/5      h264FrontEnd_CoreTaskHandler [1]
[5]       27.0         569        100       5       VBUF_New [5]
                        96          0      5/10     qurt_mutex_lock [1]
                        44          0      5/10     qurt_mutex_unlock [2]
-----------------------------------------------------------------
                       693       1141      1/1      h264FrontEnd_CoreTaskHandler [1]
[6]        6.4         104         55       1       h264BatchDecoder_TryAcquireBatchServer [6]
                        54          0      1/2      qurt_rmutex_lock [1]
                         8          0      1/2      qurt_rmutex_unlock [2]
-----------------------------------------------------------------
                       693       1141      4/4      h264FrontEnd_CoreTaskHandler [1]
[7]        5.1          93         32       4       VBUF_Release [7]
                        96          0      4/10     qurt_mutex_lock [1]
                        44          0      4/10     qurt_mutex_unlock [2]
-----------------------------------------------------------------
                       693       1141      1/1      h264FrontEnd_CoreTaskHandler [1]
[8]        4.0          16         82       1       qurt_elite_queue_pop_front [8]
                        96          0      1/10     qurt_mutex_lock [1]
                        50          0      1/1      __save_r16_through_r23 [2]
                        44          0      1/10     qurt_mutex_unlock [3]
                        27          0      1/5      qurt_anysignal_clear [4]
                        14          4      1/1      __restore_r16_through_r23_and_deallocframe [5]
-----------------------------------------------------------------
                        16         82      1/10     qurt_elite_queue_pop_front [1]
                        93         32      4/10     VBUF_Release [2]
```

Table 2-2 Values contained in the call graph

Field	Description	
`index`	Caller	Not shown
	Self	Index value of the current function For cross-reference purposes, this value is also displayed after the name on each line in the column field
	Callee	Not shown
`% time`	Caller	Not shown
	Self	Percentage of cycles that a function and its callees executed on the current thread
	Callee	Not shown
`self`	Caller	Estimated number of cycles executed by a function when called by the current caller
	Self	Total number of cycles executed by a function on the current thread
	Callee	Total number of cycles executed by the called function on the current thread
`children`	Caller	Estimated number of cycles executed by the callees of a function when called by the current caller
	Self	Total number of cycles executed by the functions called from the current function
	Callee	Total number of cycles executed by the called function's children
`called`	Caller	Total number of times the calling function called a function, followed by the total number of times the function was called
	Self	Total number of times a function was called on the current thread
	Callee	Total number of times the called function was called by the current function, followed by the total number of times the called function was called
`name`	Caller	Name of a calling function followed by its index value in brackets (such as `[1]`)
	Self	Name of a function followed by its index value in brackets
	Callee	Name of a called function followed by its index value in brackets

2.4 Profiling accuracy

The profiler can only create estimated values for the following profile data:

- Children cycle counts
- Cycle counts for function callers and callees

Because the profiler does not receive this information directly from the profile data, it must assume that for a given function F, the average number of cycles executed in each call to F is not correlated to the functions that call F.

For example, if function F1 executes for a total of 100 cycles, and one quarter of the calls to F1 originate from function F2, the profiler assumes that F1 contributes 25 cycles to F2's children cycle count.

This assumption is invalid if, for instance, F1 returns immediately after receiving a specific argument value, and F2 is the only function that ever passes that value to F1.

You are responsible for being aware of such cases and interpreting the profiling information appropriately.

3 Profile data utilities

The Hexagon SDK includes two utilities for manipulating profile data files:

- **Profile merge** – Merge profile data files
- **Profile dump** – Convert profile data file into text format

3.1 Profile merge

The profile merge utility merges two or more profile data files into a single file.

To start the profile merge utility from a command line, type:

```
hexagon-gmon-merge [-o output_file] profile_data_files...
```

The default output file is `gmon.sum`.

3.2 Profile dump

The profile dump utility converts a profile data file into a text file containing raw profiling data.

To start the profile dump utility from a command line, type:

```
hexagon-gmon-dump [-o output_file] profile_data_file
```

The default output is the standard output.

3.2.1 Dump format

A profile dump contains the following information:

- File header
- Histogram records
- Call graph records
- Basic block records

3.2.2 Example

```
File gmon.t_all:
Header:
    cookie  = 'g' 'm' 'o' 'n' ('gmon')
    version = 2

Record tag is GMON_TAG_TIME_HIST

Histogram header:
    PC range (0x00000000 - 0x00007750)
    Histogram size = 7636 (0x1dd4)
    Profiling rate = 1 (1.000000)
    Dimension = '  1.00*cyc', abbreviation = 'c'
    0x0000: 0x003a (    58)
    0x0001: 0x0000 (     0)
    0x0002: 0x0000 (     0)
.
.
.
    0x1dcf: 0x0001 (     1)
    0x1dd0: 0x0039 (    57)
    0x1dd1: 0x0002 (     2)
    0x1dd2: 0x0000 (     0)
Histogram total = 0x6d9cf4 (7183604)
Max histogram value = 0x949a5 (608677)

Record tag is GMON_TAG_CG_ARC
    ARC from 0x0000074c to 0x00005760 count = 1

Record tag is GMON_TAG_CG_ARC
    ARC from 0x00003cdc to 0x00005400 count = 1
.
.
.
Record tag is GMON_TAG_CG_ARC
    ARC from 0x00005e44 to 0x00007230 count = 1

Record tag is GMON_TAG_CG_ARC
    ARC from 0x000050d4 to 0x000070a0 count = 4

Record tag is GMON_TAG_BB_COUNT
BB record count = 1109 (0x0455)
    BB PC = 0x00000000, count =     58 (0x0000003a)
    BB PC = 0x00000098, count =     58 (0x0000003a)
    BB PC = 0x0000009c, count =      1 (0x00000001)
.
.
.
    BB PC = 0x000070c0, count =      4 (0x00000004)
    BB PC = 0x000070d0, count =      4 (0x00000004)
    BB PC = 0x000070d8, count =      4 (0x00000004)
BB Total = 0x006d9cf4 (7183604)
Max BB value = 0x949a5 (608677)
```

Table 3-1 Values contained in the profile dump

Field	Description
`cookie`	String value identifying the file format ('gmon')
`version`	File format version 1 – Standard GNU gmon format 2 – Hexagon gmon format
`Record tag`	Profile dump record tag `GMON_TAG_TIME_HIST` – Histogram record `GMON_TAG_CG_ARC` – Call graph record `GMON_TAG_BB_COUNT` – Basic block record
`PC range`	Text segment range covered by histogram
`Histogram size`	Size of histogram (in bytes) **NOTE:** Does not include header record.
`Profiling rate`	Profile clock rate
`Dimension`	Physical dimensions of bin counts (after scaling by profile clock rate)
`X: Y (  Z)`	Histogram bin X – Bin number Y – Bin size (hexadecimal value) Z – Bin size (decimal value)
`Histogram total`	Total size of histogram bins (in bytes)
`Max histogram value`	Maximum histogram value
`ARC from X to Y`	Call graph record X – Calling function address Y – Called function address
`BB record count`	Total number of basic block records
`BB PC = X, count = Y (  Z)`	X – Basic block address Y – Number of times basic block was executed (decimal value) Z – Number of times basic block was executed (hexadecimal value)
`BB Total`	Total size of basic blocks (in bytes)
`Max BB value`	Maximum basic block value