ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

How to measure the DDR throughput

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

Contents

1. How to measure the DDR throughput 3
2. DDRPERFM internal peripheral 7
3. How to stop and start Weston 13
4. Perf 15

Stable: 02.08.2021 - 06:59 / Revision: 08.07.2021 - 13:59

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

Stable: 02.08.2021 - 08:00 | Revision: 08.07.2021 - 13:00

A quality version of this page, approved on *2 August 2021*, was based off this revision.

Contents

1 Article purpose 4
2 Measurement flow 5
3 Example 6
4 References 7

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

1 Article purpose

This article explains how to measure the overall DDR throughput generated during a Linux® user space command execution.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

2 Measurement flow

The DDR throughput measurement is done via **counters** provided by the DDRPERFM internal peripheral and controlled by **stm32_ddr_pmu**[1] performance monitoring unit driver that is registered to the Linux kernel **perf framework**.

From user land side, the perf tool is used to interact with the perf framework in order to record given events (such as stm32_ddr_pmu) during a command execution.

Once recorded, the **stm32_ddr_pmu.py** Python script is used to compute the corresponding DDR throughputs.

Launching this script without any parameter returns the usage information:

```
Board $> python3 /usr/bin/stm32_ddr_pmu.py
Usage:
  python stm32_ddr_pmu.py [-d <ddr_freq>] -f <perf_file>
    -d ddr_freq: DDR frequency in MHz (533 MHz by default)
    -f perf_file: text file containing the output of
        perf stat -e stm32_ddr_pmu/read_cnt/,stm32_ddr_pmu/time_cnt/,stm32_ddr_pmu
/write_cnt/ -a -o <perf_file> <command>
```

The stm32_ddr_pmu driver is compiled as a kernel module (stm32_ddr_pmu.ko) that may not be automatically probed at startup. If 'lsmod' command does not show this module, then make sure that ddrperfm node is enabled in the device tree and run the command 'modprobe stm32_ddr_pmu' to get it properly loaded.

3 Example

Let us see how to measure the DDR throughput generated during the 'sleep 1' command execution.

First, record the counters into 'perf.txt' file, providing the command to launch as parameter:

```
Board $> perf stat -e stm32_ddr_pmu/read_cnt/,stm32_ddr_pmu/time_cnt/,stm32_ddr_pmu
/write_cnt/ -a -o perf.txt sleep 1
```

Then, run the Python script to convert the result in throughput values:

```
Board $> python3 /usr/bin/stm32_ddr_pmu.py -f perf.txt
R = 8 MB/s, W = 2 MB/s, R&W = 9 MB/s (DDR @ 533 MHz)
```

These low figures are due to the fact that the measurement was done with Weston stopped.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

4 References

- drivers/perf/stm32_ddr_pmu.c

Doubledata rate (memory domain)

Linux® is a registered trademark of Linus Torvalds.

Stable: 02.08.2021 - 06:58 / Revision: 08.07.2021 - 13:27

A quality version of this page, approved on *2 August 2021*, was based off this revision.

Contents

1 Article purpose 8
2 Peripheral overview 9
2.1 Features 9
2.2 Security support 9
3 Peripheral usage and associated software 10
3.1 Boot time 10
3.2 Runtime 10
3.2.1 Overview 10
3.2.2 Software frameworks 10
3.2.3 Peripheral configuration 10
3.2.4 Peripheral assignment 10
4 How to go further 12
5 References 13

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

1 Article purpose

The purpose of this article is to:

- briefly introduce the DDRPERFM peripheral and its main features
- indicate the level of security supported by this hardware block
- explain how it can be allocated to the three runtime contexts and linked to the corresponding software components
- explain, when necessary, how to configure the DDRPERFM peripheral.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

2 Peripheral overview

The DDRPERFM peripheral is used to count various DDRCTRL events, for performance analysis.

2.1 Features

The read, write and time counters are certainly the ones that are the most useful from user point of view, since they allow computing the DDR read and write throughputs.

Other counters are available in order to monitor the DDR controller arbitration dynamic, refresh commands and low-power management.

Refer to the STM32MP15 reference manuals for the complete list of features, and to the software components, introduced below, to see which features are implemented.

2.2 Security support

The DDRPERFM is a **non-secure** peripheral.

3 Peripheral usage and associated software

3.1 Boot time

The DDRPERFM is not used at boot time.

3.2 Runtime

3.2.1 Overview

The DDRPERFM is allocated to the Arm®Cortex®-A7 non-secure core to be controlled in Linux® by the perf framework.

Chapter Peripheral assignment describes which peripheral instance can be assigned to which context.

3.2.2 Software frameworks

Domain	Peripheral	Software components			Comment
OP-TEE	Linux	STM32Cube			
Trace & Debug	DDRPERFM		Linux perf[1] framework		

3.2.3 Peripheral configuration

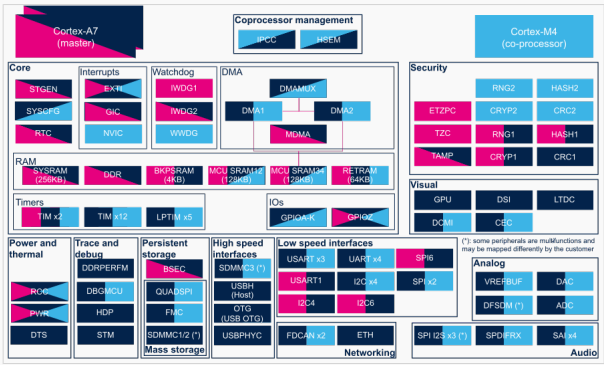
The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration can be done alone via the STM32CubeMX tool for all internal peripherals, and then manually completed (particularly for external peripherals), according to the information given in the corresponding software framework article.

3.2.4 Peripheral assignment

Check boxes illustrate the possible peripheral allocations supported by STM32 MPU Embedded Software:

- means that the peripheral can be assigned () to the given runtime context.
- is used for system peripherals that cannot be unchecked because they are statically connected in the device.

Refer to How to assign an internal peripheral to a runtime context for more information on how to assign peripherals manually or via STM32CubeMX.
The present chapter describes STMicroelectronics recommendations or choice of implementation. Additional possiblities might be described in STM32MP15 reference manuals
.



Internal peripherals

Domain	Peripheral	Runtime allocation		Comment
	Cortex-A7	Cortex-A7	Cortex-M4	

Domain	Peripheral	Runtime allocation				Comment
Instance	secure (OP-TEE)	non-secure (Linux)	(STM32Cube)			
Trace & Debug	DDRPERFM	DDRPERFM				

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

4 How to go further

Refer to How to measure the DDR throughput to learn how to use the DDRPERFM internal peripheral via the perf tool.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

5 References

- drivers/perf/stm32_ddr_pmu.c

Doubledata rate (memory domain)

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Cortex®

Linux® is a registered trademark of Linus Torvalds.

Open Portable Trusted Execution Environment

Stable: 16.03.2021 - 15:55 / Revision: 16.03.2021 - 14:22

A quality version of this page, approved on *16 March 2021*, was based off this revision.

Contents

1 Starting, stopping and restarting Weston 14
2 Preventing Weston from automatically starting on boot 15
2.1 Weston service with udev rules 15
2.2 Weston service with systemd and root user 15

1 Starting, stopping and restarting Weston

Weston has its own **systemd** service (/lib/systemd/system/weston.service) to easily start, stop and restart Weston.

The following command stops the Weston service:

```
Board $> systemctl stop weston@root.service
```

The following command starts the Weston service:

```
Board $> systemctl start weston@root.service
```

The following command stops and then restarts the Weston service:

```
Board $> systemctl restart weston@root.service
```

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

2 Preventing Weston from automatically starting on boot

2.1 Weston service with udev rules

The Weston service is automatically started on boot thanks to an **udev rules** file, most of the time located in **/etc/udev/rules.d/*weston***, for instance **/etc/udev/rules.d/71-weston-drm.rules**.

To disable this service, rename this file (by changing its extension for instance), then restart your board.

To re-enable this service, rename this file according to the udev rules of your system.

2.2 Weston service with systemd and root user

During the development process, when udev rules are not fully configured, the Weston service can be automatically started on boot as specified in **/lib/systemd/system/weston***.

In this case, the following command disables the automatic start of the Weston service:

```
Board $> systemctl disable weston@root.service
```

Note: The following command enables the automatic start of the Weston service:

```
Board $> systemctl enable weston@root.service
```

Stable: 09.10.2019 - 15:37 / Revision: 04.09.2019 - 08:00

A quality version of this page, approved on *9 October 2019*, was based off this revision.

Contents

1 Article purpose 16
2 Introduction 17
3 Installing the trace and debug tool on your target board 18
3.1 Using the STM32MPU Embedded Software distribution 18
3.2 Using the STM32MPU Embedded Software distribution for Android™ 18
4 Getting started 19
4.1 Perf commands 19
4.2 Most useful commands with simple to use interface 19
5 To go further 24
5.1 Visualizing trace using Flame Graphs 24
6 References 25

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

1 Article purpose

This article provides basic information needed to start using the Linux® kernel tool: **perf**[1].

2 Introduction

The following table provides a brief description of the tool, as well as its availability depending on the software packages:

✓: this tool is either present (ready to use or to be activated), or can be integrated and activated on the software package.

✗: this tool is not present and cannot be integrated, or it is present but cannot be activated on the software package.

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
perf	Monitoring tools	**perf**[1] is a Linux user space tool, which allows getting system performance figures	✓	✓	✓	✗*	✗*	✗*
						*Note: **simpleperf**[2] is present as equivalent but with less options*		

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

3 Installing the trace and debug tool on your target board

3.1 Using the STM32MPU Embedded Software distribution

perf is installed by default and ready to be used in all the STM32MPU Embedded Software Packages.

```
Board $> which perf
/usr/bin/perf
```

It is integrated in the weston image distribution through openembedded-core package: *openembedded-core/meta/recipes-core/packagegroups/packagegroup-core-tools-profile.bb*.

```
RRECOMMENDS_${PN} = "\
   ${PERF} \
   trace-cmd \
   blktrace \
   ${PROFILE_TOOLS_X} \
   ${PROFILE_TOOLS_SYSTEMD} \
   "
...
PERF = "perf"
```

3.2 Using the STM32MPU Embedded Software distribution for Android™

simpleperf[2] is equivalent to perf, and is installed by default (/system/xbin/simpleperf) and is ready to be used with all STM32MPU software packages for Android™.

```
Board $> which simpleperf
/system/xbin/simpleperf
```

It supports less options:

```
Board $> simpleperf --help
Usage: simpleperf [common options] subcommand [args_for_subcommand]
common options:
    -h/--help     Print this help information.
    --log <severity> Set the minimum severity of logging. Possible severities
                       include verbose, debug, warning, info, error, fatal.
                       Default is info.
    --version     Print version of simpleperf.
subcommands:
    debug-unwind        Debug/test offline unwinding.
    dump                dump perf record file
    help                print help information for simpleperf
    kmem                collect kernel memory allocation information
    list                list available event types
    record              record sampling info in perf.data
    report              report sampling information in perf.data
    report-sample       report raw sample information in perf.data
    stat                gather performance counter information
```

4 Getting started

4.1 Perf commands

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

The most commonly used perf commands are:
  annotate        Reads perf.data (created by perf record) and displays annotated code
  archive         Creates archive with object files with build-ids found in perf.data file
  bench           General framework for benchmark suites
  buildid-cache   Manages build-id cache.
  buildid-list    Lists the buildids in a perf.data file
  c2c             Shared Data C2C/HITM Analyzer.
  config          Gets and sets variables in a configuration file.
  data            Data file related processing
  diff            Reads perf.data files and displays the differential profile
  evlist          Lists the event names in a perf.data file
  ftrace          simple wrapper for kernel's ftrace functionality
  inject          Filters to augment the events stream with additional information
  kallsyms        Searches running kernel for symbols
  kmem            Tool to trace/measure kernel memory properties
  kvm             Tool to trace/measure kvm guest os
  list            Lists all symbolic event types
  lock            Analyzes lock events
  mem             Profiles memory accesses
  record          Runs a command and records its profile into perf.data
  report          Reads perf.data (created by perf record) and displays the profile
  sched           Tool to trace/measure scheduler properties (latencies)
  script          Reads perf.data (created by perf record) and displays trace output
  stat            Runs a command and gathers performance counter statistics
  test            Runs sanity tests.
  timechart       Tool to visualize total system behavior during a workload
  top             System profiling tool.
  probe           Defines new dynamic tracepoints

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

4.2 Most useful commands with simple to use interface

- **perf top** (*Linux kernel documentation*[3]): provides the CPU load by counting the number of *cycles* events; the default order is descending the number of samples per symbol:

```
Board $> perf top
 40.62%  [kernel]                          [k] v7_dma_inv_range
 18.65%  [kernel]                          [k] _raw_spin_unlock_irqrestore
 17.01%  [kernel]                          [k] arch_cpu_idle
  8.27%  [kernel]                          [k] v7_dma_clean_range
  5.00%  [kernel]                          [k] rcu_idle_exit
  1.70%  [kernel]                          [k] cpu_startup_entry
  0.52%  [kernel]                          [k] trace_graph_return
  0.48%  [kernel]                          [k] finish_task_switch
  0.48%  libc-2.18.so                      [.] memcpy
  0.47%  [kernel]                          [k] trace_graph_entry
```

Means that CPU is spending 40% of time in function v7_dma_inv_range, and 18.65% in _raw_spin_unlock_irqrestore.

More information and examples are available in perf.wiki.kernel.org[4]

This is also possible to display the result in a specified sorting:

```
Usage: perf top [<options>]
   -s, --sort <key[,key2...]>
             sort by key(s): pid, comm, dso, symbol, parent, cpu, srcline, ... Please
refer to the main page for the complete list.
```

- **perf stat** (*Linux kernel documentation*[5]): obtains event counts

```
Board $> perf stat hello_world_example
User space example: hello world from STMicroelectronics
10 9 8 7 6 5 4 3 2 1 0
User space example: goodbye from STMicroelectronics

Performance counter stats for 'hello_world_example':

          4.328249      task-clock (msec)         #    0.000 CPUs utilized
                11      context-switches          #    0.003 M/sec
                 0      cpu-migrations            #    0.000 K/sec
                38      page-faults               #    0.009 M/sec
           2710036      cycles                    #    0.626 GHz
            640856      instructions              #    0.24  insn per cycle
             75644      branches                  #   17.477 M/sec
             21764      branch-misses             #   28.77% of all branches

      11.109859338 seconds time elapsed
```

More information and examples are available in perf.wiki.kernel.org[6].

- **perf list** (*Linux kernel documentation*[7]): supported symbolic event types

```
Board $> perf list
 branch-instructions OR branches                    [Hardware event]
 branch-misses                                      [Hardware event]
 bus-cycles                                         [Hardware event]
 cache-misses                                       [Hardware event]
 cache-references                                   [Hardware event]
 cpu-cycles OR cycles                               [Hardware event]
 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]
 L1-dcache-store-misses                             [Hardware cache event]
 L1-dcache-stores                                   [Hardware cache event]
 L1-icache-load-misses                              [Hardware cache event]
 L1-icache-loads                                    [Hardware cache event]
```

```
  LLC-load-misses                                    [Hardware cache event]
  LLC-loads                                          [Hardware cache event]
  LLC-store-misses                                   [Hardware cache event]
  LLC-stores                                         [Hardware cache event]
  branch-load-misses                                 [Hardware cache event]
  branch-loads                                       [Hardware cache event]
  dTLB-load-misses                                   [Hardware cache event]
  dTLB-store-misses                                  [Hardware cache event]
  iTLB-load-misses                                   [Hardware cache event]
  armv7_cortex_a7/br_immed_retired/                  [Kernel PMU event]
  armv7_cortex_a7/br_mis_pred/                       [Kernel PMU event]
  armv7_cortex_a7/br_pred/                           [Kernel PMU event]
  armv7_cortex_a7/br_return_retired/                 [Kernel PMU event]
  armv7_cortex_a7/bus_access/                        [Kernel PMU event]
  armv7_cortex_a7/bus_cycles/                        [Kernel PMU event]
  armv7_cortex_a7/cid_write_retired/                 [Kernel PMU event]
  armv7_cortex_a7/cpu_cycles/                        [Kernel PMU event]
  armv7_cortex_a7/exc_return/                        [Kernel PMU event]
  armv7_cortex_a7/exc_taken/                         [Kernel PMU event]
  armv7_cortex_a7/inst_retired/                      [Kernel PMU event]
  armv7_cortex_a7/inst_spec/                         [Kernel PMU event]
  armv7_cortex_a7/l1d_cache/                         [Kernel PMU event]
  armv7_cortex_a7/l1d_cache_refill/                  [Kernel PMU event]
  armv7_cortex_a7/l1d_cache_wb/                      [Kernel PMU event]
  armv7_cortex_a7/l1d_tlb_refill/                    [Kernel PMU event]
  armv7_cortex_a7/l1i_cache/                         [Kernel PMU event]
  armv7_cortex_a7/l1i_cache_refill/                  [Kernel PMU event]
  armv7_cortex_a7/l1i_tlb_refill/                    [Kernel PMU event]
  armv7_cortex_a7/l2d_cache/                         [Kernel PMU event]
  armv7_cortex_a7/l2d_cache_refill/                  [Kernel PMU event]
  armv7_cortex_a7/l2d_cache_wb/                      [Kernel PMU event]
  armv7_cortex_a7/ld_retired/                        [Kernel PMU event]
  armv7_cortex_a7/mem_access/                        [Kernel PMU event]
  armv7_cortex_a7/memory_error/                      [Kernel PMU event]
  armv7_cortex_a7/pc_write_retired/                  [Kernel PMU event]
  armv7_cortex_a7/st_retired/                        [Kernel PMU event]
  armv7_cortex_a7/sw_incr/                           [Kernel PMU event]
  armv7_cortex_a7/ttbr_write_retired/                [Kernel PMU event]
  armv7_cortex_a7/unaligned_ldst_retired/            [Kernel PMU event]
  rNNN                                               [Raw hardware event descriptor]
  cpu/t1=v1[,t2=v2,t3 ...]/modifier                  [Raw hardware event descriptor]
  mem:<addr>[/len][:access]                          [Hardware breakpoint]
  alarmtimer:alarmtimer_cancel                       [Tracepoint event]
  alarmtimer:alarmtimer_fired                        [Tracepoint event]
  alarmtimer:alarmtimer_start                        [Tracepoint event]
  alarmtimer:alarmtimer_suspend                      [Tracepoint event]
  asoc:snd_soc_bias_level_done                       [Tracepoint event]
  asoc:snd_soc_bias_level_start                      [Tracepoint event]
  asoc:snd_soc_dapm_connected                        [Tracepoint event]
  asoc:snd_soc_dapm_done                             [Tracepoint event]
  asoc:snd_soc_dapm_path                             [Tracepoint event]
  asoc:snd_soc_dapm_start                            [Tracepoint event]
  asoc:snd_soc_dapm_walk_done                        [Tracepoint event]
  asoc:snd_soc_dapm_widget_event_done                [Tracepoint event]
  asoc:snd_soc_dapm_widget_event_start               [Tracepoint event]
...
  xhci-hcd:xhci_inc_enq                              [Tracepoint event]
  xhci-hcd:xhci_queue_trb                            [Tracepoint event]
  xhci-hcd:xhci_ring_alloc                           [Tracepoint event]
  xhci-hcd:xhci_ring_expansion                       [Tracepoint event]
  xhci-hcd:xhci_ring_free                            [Tracepoint event]
  xhci-hcd:xhci_setup_addressable_virt_device        [Tracepoint event]
  xhci-hcd:xhci_setup_device                         [Tracepoint event]
```

```
  xhci-hcd:xhci_setup_device_slot                    [Tracepoint event]
  xhci-hcd:xhci_stop_device                          [Tracepoint event]
  xhci-hcd:xhci_urb_dequeue                          [Tracepoint event]
  xhci-hcd:xhci_urb_enqueue                          [Tracepoint event]
  xhci-hcd:xhci_urb_giveback                         [Tracepoint event]
```

- **perf record** (*Linux kernel documentation*[8]): records events for later reporting

```
Board $> perf record hello_world_example

User space example: hello world from STMicroelectronics
10 9 8 7 6 5 4 3 2 1 0
User space example: goodbye from STMicroelectronics
[ perf record: Woken up 1 time to write data ]
[ perf record: Captured and wrote 0.004 MB perf.data (28 samples) ]
```

This is possible to filter events (given by *perf list* command). More information, options and examples are available in perf.wiki.kernel.org[9].

By default, the events are recorded in the ***perf.data*** file. If you want to specify another output file name you have to add *-o, --output <file>* option.

- **perf report** (*Linux kernel documentation*[10]): breaks down the events by process, function, etc.

```
Example after previous command "perf record hello_world_example"
Board $> perf report
Samples: 28  of event 'cycles:ppp', Event count (approx.):2737925
Overhead  Command          Shared Object Symbol
  12.66%  hello_world_exa  ld-2.26.so         [.] _dl_relocate_object
  11.71%  hello_world_exa  [kernel.kallsyms]  [k] filemap_map_pages
  10.65%  hello_world_exa  [kernel.kallsyms]  [k] n_tty_write
   6.43%  hello_world_exa  [kernel.kallsyms]  [k] percpu_counter_add_batch
   6.43%  hello_world_exa  ld-2.26.so         [.] sbrk
   6.24%  hello_world_exa  [kernel.kallsyms]  [k] cpu_v7_set_pte_ext
   5.56%  hello_world_exa  [kernel.kallsyms]  [k] alloc_set_pte
   5.56%  hello_world_exa  libc-2.26.so       [.] __sbrk
   5.37%  hello_world_exa  [kernel.kallsyms]  [k] __vma_link_file
   5.32%  hello_world_exa  [kernel.kallsyms]  [k] __fput
   5.32%  hello_world_exa  [kernel.kallsyms]  [k] ldsem_up_read
   5.32%  hello_world_exa  [kernel.kallsyms]  [k] unmap_page_range
   5.32%  hello_world_exa  libc-2.26.so       [.] printf
   5.24%  hello_world_exa  [kernel.kallsyms]  [k] _raw_spin_unlock_irqrestore
   2.23%  hello_world_exa  [kernel.kallsyms]  [k] perf_event_mmap
   0.48%  hello_world_exa  [kernel.kallsyms]  [k] perf_output_begin
   0.13%  perf             [kernel.kallsyms]  [k] perf_event_exec
```

By default, report file ***perf.data*** is read as input file. If you want to specify another input file name you have to add *-i, --input <file>* option.

More information and examples are available in perf.wiki.kernel.org[11].

- **perf bench** (*Linux kernel documentation*[12]): runs different kernel microbenchmarks:

```
# List of all available benchmark collections:

        sched: Scheduler and IPC benchmarks
          mem: Memory access benchmarks
        futex: Futex stressing benchmarks
          all: All benchmarks
```

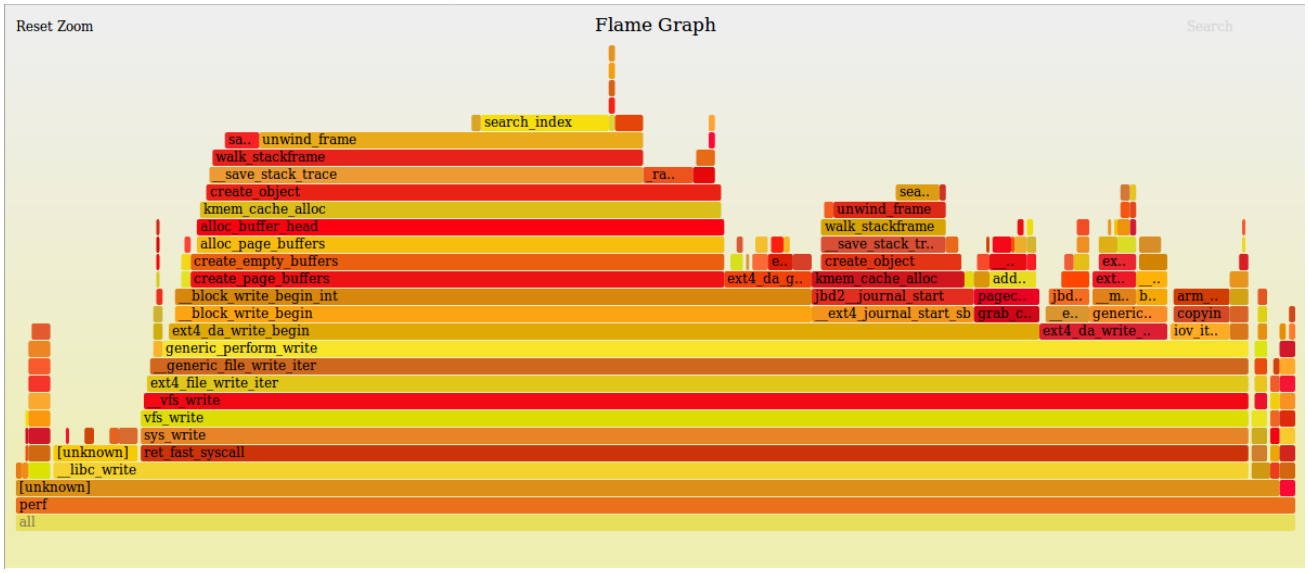
```
Example of getting memcpy benchmark for 100MB:
Board $> perf bench mem memcpy --size 100MB
# Running 'mem/memcpy' benchmark:
# function 'default' (Default memcpy() provided by glibc)
# Copying 100MB bytes ...

       1.426138 GB/sec
```

More information and examples are available in perf.wiki.kernel.org[13].

5 To go further

5.1 Visualizing trace using Flame Graphs

As part of Flame Graphs[14], this is possible to visualize trace coming from perf.



The Flame graphs are generated using Flame graphs tool suite[15].

- Install the Flame Graph tool suite on host PC side

```
PC $> cd <your_local_path>
PC $> git clone https://github.com/brendangregg/FlameGraph.git
PC $> cd FlameGraph
```

- Generate a Flame graph from perf tool

  When generating perf record, ***-g* option must be added**.

As example for a *top* command:

```
- Perform perf record command on board side
Board $> perf record -a -g top
Board $> perf script > perf_top.out

- Copy perf_top.out on your host PC (i.e. in the FlameGraph directory)

- Perform the flame graph generation on host PC side using stackcollapse-perf.pl script
PC $> ./stackcollapse-perf.pl perf_top.out > out.top_folded

- Use flamegraph.pl to render a SVG (Scalable Vector Graphics) file.
PC $> ./flamegraph.pl out.top_folded > top.svg

- Visualize SVG using web browser for example
PC $> firefox top.svg
```

6 References

- [1.01.1] https://perf.wiki.kernel.org/index.php/Main_Page
- [2.02.1] https://source.android.com/devices/tech/debug/eval_perf
- tools/perf/Documentation/perf-top.txt
- https://perf.wiki.kernel.org/index.php/Tutorial#Live_analysis_with_perf_top
- tools/perf/Documentation/perf-stat.txt
- https://perf.wiki.kernel.org/index.php/Tutorial#Counting_with_perf_stat
- tools/perf/Documentation/perf-list.txt
- tools/perf/Documentation/perf-record.txt
- https://perf.wiki.kernel.org/index.php/Tutorial#Sampling_with_perf_record
- tools/perf/Documentation/perf-report.txt
- https://perf.wiki.kernel.org/index.php/Tutorial#Sample_analysis_with_perf_report
- tools/perf/Documentation/perf-bench.txt
- https://perf.wiki.kernel.org/index.php/Tutorial#Benchmarking_with_perf_bench
- http://www.brendangregg.com/flamegraphs.html
- https://github.com/brendangregg/FlameGraph

- Useful external links

Document link	Document Type	Description
perf tutorial	User Guide	perf.wiki.kernel.org
perf (wikipedia.org)	Standard	wikipedia.org
Brendan Gregg's perf page	Perf example	From Brendan Gregg
Eclipse perf plugin page	Eclipse perf plugin	Eclipse.org

Linux® is a registered trademark of Linus Torvalds.

Central processing unit

Power Management Unit (in STPMIC context) or Performance Monitoring Unit (in Arm® Cortex®-A context)

Inter-Processor Communication