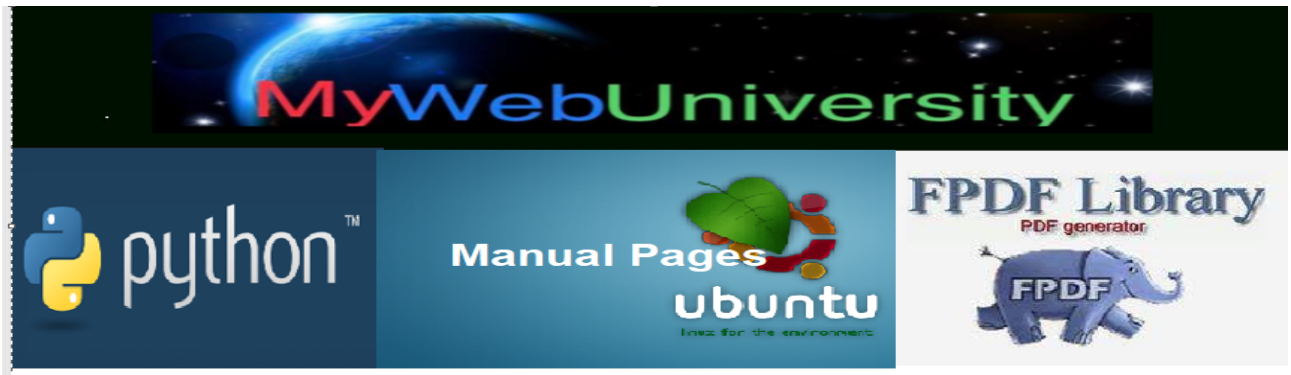

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'perf-stat.1'

$ man perf-stat.1

PERF-STAT(1) perf Manual PERF-STAT(1)

NAME

perf-stat - Run a command and gather performance counter statistics

SYNOPSIS

perf stat [-e <EVENT> | --event=EVENT] [-a] <command>

perf stat [-e <EVENT> | --event=EVENT] [-a] -- <command> [<options>]

perf stat [-e <EVENT> | --event=EVENT] [-a] record [-o file] -- <command> [<options>]

perf stat report [-i file]

DESCRIPTION

This command runs a command and gathers performance counter statistics from it.

OPTIONS

<command>...

Any command you can specify in a shell.

record

See STAT RECORD.

report

See STAT REPORT.

-e, --event=

Select the PMU event. Selection can be:

? a symbolic event name (use perf list to list all events)

? a raw PMU event in the form of rN where N is a hexadecimal value that represents the raw register encoding with the layout of the event control registers as described by entries in /sys/bus/event_source/devices/cpu/format/*.

? a symbolic or raw PMU event followed by an optional colon and a list of event modifiers, e.g., cpu-cycles:p. See the perf-list(1) man page for details on event modifiers.

? a symbolically formed event like pmu/param1=0x3,param2/ where param1 and param2 are defined as formats for the PMU in /sys/bus/event_source/devices/<pmu>/format/*

'percore' is a event qualifier that sums up the event counts for both hardware threads in a core. For example:

perf stat -A -a -e cpu/event,percore=1/,otherevent ...

? a symbolically formed event like pmu/config=M,config1=N,config2=K/ where M, N, K are numbers (in decimal, hex, octal format). Acceptable values for each of config, config1 and config2 parameters are defined by corresponding entries in /sys/bus/event_source/devices/<pmu>/format/*

Note that the last two syntaxes support prefix and glob matching in the PMU name to simplify creation of events across multiple instances of the same type of PMU in large systems (e.g. memory controller PMUs). Multiple PMU instances are typical for uncore PMUs, so the prefix 'uncore_' is also ignored when performing this match.

-i, --no-inherit

child tasks do not inherit counters

-p, --pid=<pid>

stat events on existing process id (comma separated list)

-t, --tid=<tid>

stat events on existing thread id (comma separated list)

-b, --bpf-prog

stat events on existing bpf program id (comma separated list), requiring root rights. bpftool-prog could be used to find program id all bpf programs in the system. For example:

```
# bpftool prog | head -n 1
17247: tracepoint  name sys_enter  tag 192d548b9d754067  gpl
# perf stat -e cycles,instructions --bpf-prog 17247 --timeout 1000
Performance counter stats for 'BPF program(s) 17247':
85,967      cycles
28,982      instructions              #    0.34  insn per cycle
```

1.102235068 seconds time elapsed

--bpf-counters

Use BPF programs to aggregate readings from perf_events. This allows multiple perf-stat sessions that are counting the same metric (cycles, instructions, etc.) to share hardware counters. To use BPF programs on common events by default, use "perf config stat.bpf-counter-events=<list_of_events>".

--bpf-attr-map

With option "--bpf-counters", different perf-stat sessions share information about shared BPF programs and maps via a pinned hashmap. Use "--bpf-attr-map" to specify the path of this pinned hashmap. The default path is /sys/fs/bpf/perf_attr_map.

-a, --all-cpus

system-wide collection from all CPUs (default if no target is specified)

--no-scale

Don?t scale/normalize counter values

-d, --detailed

print more detailed statistics, can be specified up to 3 times

-d: detailed events, L1 and LLC data cache

-d -d: more detailed events, dTLB and iTLB events

-d -d -d: very detailed events, adding prefetch events

-r, --repeat=<n>

repeat command and print average + stddev (max: 100). 0 means forever.

-B, --big-num

print large numbers with thousands' separators according to locale. Enabled by default. Use "--no-big-num" to disable. Default setting can be changed with "perf config stat.big-num=false".

-C, --cpu=

Count only on the list of CPUs provided. Multiple CPUs can be provided as a comma-separated list with no space: 0,1. Ranges of CPUs are specified with -: 0-2. In per-thread mode, this option is ignored. The -a option is still necessary to activate system-wide monitoring. Default is to count on all CPUs.

-A, --no-aggr

Do not aggregate counts across all monitored CPUs.

-n, --null

null run - Don?t start any counters.

This can be useful to measure just elapsed wall-clock time - or to assess the raw overhead of perf stat itself, without running any counters.

-v, --verbose

be more verbose (show counter open errors, etc)

-x SEP, --field-separator SEP

print counts using a CSV-style output to make it easy to import directly into spreadsheets. Columns are separated by the string specified in SEP.

--table

Display time for each run (-r option), in a table format, e.g.:

```
$ perf stat --null -r 5 --table perf bench sched pipe

Performance counter stats for 'perf bench sched pipe' (5 runs):

# Table of individual measurements:

5.189 (-0.293) #

5.189 (-0.294) #

5.186 (-0.296) #

5.663 (+0.181) ##

6.186 (+0.703) ####

# Final result:

5.483 +- 0.198 seconds time elapsed  ( +-  3.62% )
```

-G name, --cgroup name

monitor only in the container (cgroup) called "name". This option is available only in per-cpu mode. The cgroup filesystem must be mounted. All threads belonging to container "name" are monitored when they run on the monitored CPUs. Multiple cgroups can be provided. Each cgroup is applied to the corresponding event, i.e., first cgroup to first event, second cgroup to second event and so on. It is possible to provide an empty cgroup (monitor all the time) using, e.g., -G foo,,bar. Cgroups must have corresponding events, i.e., they always refer to events defined earlier on the command line. If the user wants to track multiple events for a specific cgroup, the user can use -e e1 -e e2 -G foo,foo or just use -e e1 -e e2 -G foo.

If wanting to monitor, say, cycles for a cgroup and also for system wide, this command line can be used: perf stat -e cycles -G cgroup_name -a -e cycles.

--for-each-cgroup name

Expand event list for each cgroup in "name" (allow multiple cgroups separated by comma). It also support regex patterns to match multiple groups. This has same effect that repeating -e option and -G option for each event x name. This option cannot be used with -G/--cgroup option.

-o file, --output file

Print the output into the designated file.

--append

Append to the output file designated with the -o option. Ignored if -o is not specified.

--log-fd

Log output to fd, instead of stderr. Complementary to --output, and mutually exclusive with it. --append may be used here. Examples: 3>results perf stat --log-fd 3 -- $cmd 3>>results perf stat --log-fd 3 --append -- $cmd

--control=fifo:ctl-fifo[,ack-fifo], --control=fd:ctl-fd[,ack-fd]

ctl-fifo / ack-fifo are opened and used as ctl-fd / ack-fd as follows. Listen on ctl-fd descriptor for command to control measurement (enable: enable events, disable: disable events). Measurements can be started with events disabled using --delay=-1 option. Optionally send control command completion (ack\n) to ack-fd descriptor to synchronize with the controlling process. Example of bash shell script to enable and disable events during measurements:

```
#!/bin/bash
ctl_dir=/tmp/
ctl_fifo=${ctl_dir}perf_ctl.fifo
test -p ${ctl_fifo} && unlink ${ctl_fifo}
mkfifo ${ctl_fifo}
exec {ctl_fd}<>${ctl_fifo}
ctl_ack_fifo=${ctl_dir}perf_ctl_ack.fifo
test -p ${ctl_ack_fifo} && unlink ${ctl_ack_fifo}
mkfifo ${ctl_ack_fifo}
exec {ctl_fd_ack}<>${ctl_ack_fifo}
perf stat -D -1 -e cpu-cycles -a -I 1000        \
        --control fd:${ctl_fd},${ctl_fd_ack} \
        \-- sleep 30 &
perf_pid=$!
```

```
        sleep 5  && echo 'enable' >&${ctl_fd} && read -u ${ctl_fd_ack} e1 && echo "enabled(${e1})"
        sleep 10 && echo 'disable' >&${ctl_fd} && read -u ${ctl_fd_ack} d1 && echo "disabled(${d1})"
        exec {ctl_fd_ack}>&-
        unlink ${ctl_ack_fifo}
        exec {ctl_fd}>&-
        unlink ${ctl_fifo}
        wait -n ${perf_pid}
        exit $?
```

--pre, --post

Pre and post measurement hooks, e.g.:

perf stat --repeat 10 --null --sync --pre make -s O=defconfig-build/clean -- make -s -j64 O=defconfig-build/ bzImage

-I msecs, --interval-print msecs

Print count deltas every N milliseconds (minimum: 1ms) The overhead percentage could be high in some cases, for instance with small, sub 100ms intervals. Use with caution.

example: perf stat -I 1000 -e cycles -a sleep 5

If the metric exists, it is calculated by the counts generated in this interval and the metric is printed after #.

--interval-count times

Print count deltas for fixed number of times. This option should be used together with "-I" option. example: perf stat -I 1000 --interval-count 2 -e cycles -a

--interval-clear

Clear the screen before next interval.

--timeout msecs

Stop the perf stat session and print count deltas after N milliseconds (minimum: 10 ms). This option is not supported with the "-I" option. example: perf stat --time 2000 -e cycles -a

--metric-only

Only print computed metrics. Print them in a single line. Don?t show any raw values. Not supported with --per-thread.

--per-socket

Aggregate counts per processor socket for system-wide mode measurements. This is a useful mode to detect imbalance between sockets. To enable this mode, use --per-socket

in addition to -a. (system-wide). The output includes the socket number and the number of online processors on that socket. This is useful to gauge the amount of aggregation.

--per-die

Aggregate counts per processor die for system-wide mode measurements. This is a useful mode to detect imbalance between dies. To enable this mode, use --per-die in addition to -a. (system-wide). The output includes the die number and the number of online processors on that die. This is useful to gauge the amount of aggregation.

--per-cluster

Aggregate counts per processor cluster for system-wide mode measurement. This is a useful mode to detect imbalance between clusters. To enable this mode, use --per-cluster in addition to -a. (system-wide). The output includes the cluster number and the number of online processors on that cluster. This is useful to gauge the amount of aggregation. The information of cluster ID and related CPUs can be gotten from /sys/devices/system/cpu/cpuX/topology/cluster_{id, cpus}.

--per-cache

Aggregate counts per cache instance for system-wide mode measurements. By default, the aggregation happens for the cache level at the highest index in the system. To specify a particular level, mention the cache level alongside the option in the format [Ll][1-9][0-9]*. For example: Using option "--per-cache=l3" or "--per-cache=L3" will aggregate the information at the boundary of the level 3 cache in the system.

--per-core

Aggregate counts per physical processor for system-wide mode measurements. This is a useful mode to detect imbalance between physical cores. To enable this mode, use --per-core in addition to -a. (system-wide). The output includes the core number and the number of online logical processors on that physical processor.

--per-thread

Aggregate counts per monitored threads, when monitoring threads (-t option) or processes (-p option).

--per-node

Aggregate counts per NUMA nodes for system-wide mode measurements. This is a useful mode to detect imbalance between NUMA nodes. To enable this mode, use --per-node in addition to -a. (system-wide).

-D msecs, --delay msecs

After starting the program, wait msecs before measuring (-1: start with events disabled). This is useful to filter out the startup phase of the program, which is often very different.

-T, --transaction

Print statistics of transactional execution if supported.

--metric-no-group

By default, events to compute a metric are placed in weak groups. The group tries to enforce scheduling all or none of the events. The --metric-no-group option places events outside of groups and may increase the chance of the event being scheduled - leading to more accuracy. However, as events may not be scheduled together accuracy for metrics like instructions per cycle can be lower - as both metrics may no longer be being measured at the same time.

--metric-no-merge

By default metric events in different weak groups can be shared if one group contains all the events needed by another. In such cases one group will be eliminated reducing event multiplexing and making it so that certain groups of metrics sum to 100%. A downside to sharing a group is that the group may require multiplexing and so accuracy for a small group that need not have multiplexing is lowered. This option forbids the event merging logic from sharing events between groups and may be used to increase accuracy in this case.

--metric-no-threshold

Metric thresholds may increase the number of events necessary to compute whether a metric has exceeded its threshold expression. This may not be desirable, for example, as the events can introduce multiplexing. This option disables the adding of threshold expression events for a metric. However, if there are sufficient events to compute the threshold then the threshold is still computed and used to color the metric?s computed value.

--quiet

Don?t print output, warnings or messages. This is useful with perf stat record below to only write data to the perf.data file.

--no-affinity

Don?t change scheduler CPU affinities when iterating over CPUs. Disables an optimization aimed at minimizing interprocessor interrupts.

STAT RECORD

Stores stat data into perf data file.

-o file, --output file

Output file name.

STAT REPORT

Reads and reports stat data from perf data file.

-i file, --input file

Input file name.

--per-socket

Aggregate counts per processor socket for system-wide mode measurements.

--per-die

Aggregate counts per processor die for system-wide mode measurements.

--per-cluster

Aggregate counts perf processor cluster for system-wide mode measurements.

--per-cache

Aggregate counts per cache instance for system-wide mode measurements. By default, the aggregation happens for the cache level at the highest index in the system. To specify a particular level, mention the cache level alongside the option in the format [Ll][1-9][0-9]*. For example: Using option "--per-cache=l3" or "--per-cache=L3" will aggregate the information at the boundary of the level 3 cache in the system.

--per-core

Aggregate counts per physical processor for system-wide mode measurements.

-M, --metrics

Print metrics or metricgroups specified in a comma separated list. For a group all metrics from the group are added. The events from the metrics are automatically measured. See perf list output for the possible metrics and metricgroups.

When threshold information is available for a metric, the color red is used to signify a metric has exceeded a threshold while green shows it hasn't. The default color means that no threshold information was available or the threshold couldn't be computed.

-A, --no-aggr, --no-merge

Do not aggregate/merge counts across monitored CPUs or PMUs.

When multiple events are created from a single event specification, stat will, by default, aggregate the event counts and show the result in a single row. This option disables that behavior and shows the individual events and counts.

Multiple events are created from a single event specification when:

1. PID monitoring isn?t requested and the system has more than one CPU. For example, a system with 8 SMT threads will have one event opened on each thread and aggregation is performed across them.
2. Prefix or glob wildcard matching is used for the PMU name. For example, multiple memory controller PMUs may exist typically with a suffix of _0, _1, etc. By default the event counts will all be combined if the PMU is specified without the suffix such as uncore_imc rather than uncore_imc_0.
3. Aliases, which are listed immediately after the Kernel PMU events by perf list, are used.

--hybrid-merge

Merge core event counts from all core PMUs. In hybrid or big.LITTLE systems by default each core PMU will report its count separately. This option forces core PMU counts to be combined to give a behavior closer to having a single CPU type in the system.

--topdown

Print top-down metrics supported by the CPU. This allows to determine bottle necks in the CPU pipeline for CPU bound workloads, by breaking the cycles consumed down into frontend bound, backend bound, bad speculation and retiring.

Frontend bound means that the CPU cannot fetch and decode instructions fast enough. Backend bound means that computation or memory access is the bottle neck. Bad Speculation means that the CPU wasted cycles due to branch mispredictions and similar issues. Retiring means that the CPU computed without an apparently bottleneck. The bottleneck is only the real bottleneck if the workload is actually bound by the CPU and not by something else.

For best results it is usually a good idea to use it with interval mode like -I 1000, as the bottleneck of workloads can change often.

This enables --metric-only, unless overridden with --no-metric-only.

The following restrictions only apply to older Intel CPUs and Atom, on newer CPUs (IceLake and later) TopDown can be collected for any thread:

The top down metrics are collected per core instead of per CPU thread. Per core mode is

automatically enabled and -a (global monitoring) is needed, requiring root rights or perf.perf_event_paranoid=-1.

Topdown uses the full Performance Monitoring Unit, and needs disabling of the NMI watchdog (as root): echo 0 > /proc/sys/kernel/nmi_watchdog for best results. Otherwise the bottlenecks may be inconsistent on workload with changing phases.

To interpret the results it is usually needed to know on which CPUs the workload runs on. If needed the CPUs can be forced using taskset.

--record-tpebs

Enable automatic sampling on Intel TPEBS retire_latency events (event with :R modifier). Without this option, perf would not capture dynamic retire_latency at runtime. Currently, a zero value is assigned to the retire_latency event when this option is not set. The TPEBS hardware feature starts from Intel Granite Rapids microarchitecture. This option only exists in X86_64 and is meaningful on Intel platforms with TPEBS feature.

--tpebs-mode=[mean|min|max|last]

Set how retirement latency events have their sample times combined. The default "mean" gives the average of retirement latency. "min" or "max" give the smallest or largest retirment latency times respectively. "last" uses the last retirment latency sample?s time.

--td-level

Print the top-down statistics that equal the input level. It allows users to print the interested top-down metrics level instead of the level 1 top-down metrics.

As the higher levels gather more metrics and use more counters they will be less accurate. By convention a metric can be examined by appending _group to it and this will increase accuracy compared to gathering all metrics for a level. For example, level 1 analysis may highlight tma_frontend_bound. This metric may be drilled into with tma_frontend_bound_group with perf stat -M tma_frontend_bound_group....

Error out if the input is higher than the supported max level.

--smi-cost

Measure SMI cost if msr/aperf/ and msr/smi/ events are supported.

During the measurement, the /sys/device/cpu/freeze_on_smi will be set to freeze core counters on SMI. The aperf counter will not be effected by the setting. The cost of SMI can be measured by (aperf - unhalted core cycles).

In practice, the percentages of SMI cycles is very useful for performance oriented analysis.

--metric_only will be applied by default. The output is SMI cycles%, equals to (aperf - unhalted core cycles) / aperf

Users who wants to get the actual value can apply --no-metric-only.

--all-kernel

Configure all used events to run in kernel space.

--all-user

Configure all used events to run in user space.

--percore-show-thread

The event modifier "percore" has supported to sum up the event counts for all hardware threads in a core and show the counts per core.

This option with event modifier "percore" enabled also sums up the event counts for all hardware threads in a core but show the sum counts per hardware thread. This is essentially a replacement for the any bit and convenient for post processing.

--summary

Print summary for interval mode (-I).

--no-csv-summary

Don?t print summary at the first column for CVS summary output. This option must be used with -x and --summary.

This option can be enabled in perf config by setting the variable stat.no-csv-summary.

$ perf config stat.no-csv-summary=true

--cputype

Only enable events on applying cpu with this type for hybrid platform (e.g. core or atom)"

EXAMPLES

$ perf stat -- make

Performance counter stats for 'make':

```
  83723.452481      task-clock:u (msec)       #    1.004 CPUs utilized
           0      context-switches:u        #    0.000 K/sec
           0      cpu-migrations:u          #    0.000 K/sec
       3,228,188      page-faults:u             #    0.039 M/sec
 229,570,665,834      cycles:u                  #    2.742 GHz
 313,163,853,778      instructions:u            #    1.36  insn per cycle
  69,704,684,856      branches:u                #  832.559 M/sec
```

2,078,861,393 branch-misses:u # 2.98% of all branches

83.409183620 seconds time elapsed

74.684747000 seconds user

8.739217000 seconds sys

TIMINGS

As displayed in the example above we can display 3 types of timings. We always display the time the counters were enabled/alive:

83.409183620 seconds time elapsed

For workload sessions we also display time the workloads spent in user/system lands:

74.684747000 seconds user

8.739217000 seconds sys

Those times are the very same as displayed by the time tool.

CSV FORMAT

With -x, perf stat is able to output a not-quite-CSV format output Commas in the output are not put into "". To make it easy to parse it is recommended to use a different character like -x \;

The fields are in this order:

? optional usec time stamp in fractions of second (with -I xxx)

? optional CPU, core, or socket identifier

? optional number of logical CPUs aggregated

? counter value

? unit of the counter value or empty

? event name

? run time of counter

? percentage of measurement time the counter was running

? optional variance if multiple values are collected with -r

? optional metric value

? optional unit of metric

Additional metrics may be printed with all earlier fields being empty.

INTEL HYBRID SUPPORT

Support for Intel hybrid events within perf tools.

For some Intel platforms, such as AlderLake, which is hybrid platform and it consists of atom cpu and core cpu. Each cpu has dedicated event list. Part of events are available on

core cpu, part of events are available on atom cpu and even part of events are available on both.

Kernel exports two new cpu pmus via sysfs: /sys/bus/event_source/devices/cpu_core /sys/bus/event_source/devices/cpu_atom

The cpus files are created under the directories. For example,

cat /sys/bus/event_source/devices/cpu_core/cpus 0-15

cat /sys/bus/event_source/devices/cpu_atom/cpus 16-23

It indicates cpu0-cpu15 are core cpus and cpu16-cpu23 are atom cpus.

As before, use perf-list to list the symbolic event.

perf list

inst_retired.any [Fixed Counter: Counts the number of instructions retired. Unit: cpu_atom]

inst_retired.any [Number of instructions retired. Fixed Counter - architectural event. Unit: cpu_core]

The Unit: xxx is added to brief description to indicate which pmu the event is belong to.

Same event name but with different pmu can be supported.

Enable hybrid event with a specific pmu

To enable a core only event or atom only event, following syntax is supported:

cpu_core/<event name>/

or

cpu_atom/<event name>/

For example, count the cycles event on core cpus.

perf stat -e cpu_core/cycles/

Create two events for one hardware event automatically

When creating one event and the event is available on both atom and core, two events are created automatically. One is for atom, the other is for core. Most of hardware events and cache events are available on both cpu_core and cpu_atom.

For hardware events, they have pre-defined configs (e.g. 0 for cycles). But on hybrid platform, kernel needs to know where the event comes from (from atom or from core). The original perf event type PERF_TYPE_HARDWARE can?t carry pmu information. So now this type is extended to be PMU aware type. The PMU type ID is stored at attr.config[63:32].

PMU type ID is retrieved from sysfs. /sys/bus/event_source/devices/cpu_atom/type /sys/bus/event_source/devices/cpu_core/type

The new attr.config layout for PERF_TYPE_HARDWARE:

PERF_TYPE_HARDWARE: 0xEEEEEEEE000000AA AA: hardware event ID EEEEEEEE: PMU type ID

Cache event is similar. The type PERF_TYPE_HW_CACHE is extended to be PMU aware type. The PMU type ID is stored at attr.config[63:32].

The new attr.config layout for PERF_TYPE_HW_CACHE:

PERF_TYPE_HW_CACHE: 0xEEEEEEEE00DDCCBB BB: hardware cache ID CC: hardware cache op ID DD: hardware cache op result ID EEEEEEEE: PMU type ID

When enabling a hardware event without specified pmu, such as, perf stat -e cycles -a (use system-wide in this example), two events are created automatically.

```
    ---------------------------------------------------------
    perf_event_attr:
      size                    120
      config                  0x400000000
      sample_type             IDENTIFIER
      read_format             TOTAL_TIME_ENABLED|TOTAL_TIME_RUNNING
      disabled                1
      inherit                 1
      exclude_guest           1
    ---------------------------------------------------------
```

and

```
    ---------------------------------------------------------
    perf_event_attr:
      size                    120
      config                  0x800000000
      sample_type             IDENTIFIER
      read_format             TOTAL_TIME_ENABLED|TOTAL_TIME_RUNNING
      disabled                1
      inherit                 1
      exclude_guest           1
    ---------------------------------------------------------
```

type 0 is PERF_TYPE_HARDWARE. 0x4 in 0x400000000 indicates it?s cpu_core pmu. 0x8 in 0x800000000 indicates it?s cpu_atom pmu (atom pmu type id is random).

The kernel creates cycles (0x400000000) on cpu0-cpu15 (core cpus), and create cycles (0x800000000) on cpu16-cpu23 (atom cpus).

For perf-stat result, it displays two events:

Performance counter stats for 'system wide':

6,744,979 cpu_core/cycles/

1,965,552 cpu_atom/cycles/

The first cycles is core event, the second cycles is atom event.

Thread mode example:

perf-stat reports the scaled counts for hybrid event and with a percentage displayed. The percentage is the event?s running time/enabling time.

One example, triad_loop runs on cpu16 (atom core), while we can see the scaled value for core cycles is 160,444,092 and the percentage is 0.47%.

perf stat -e cycles -- taskset -c 16 ./triad_loop

As previous, two events are created.

```
.ft C
perf_event_attr:
  size                    120
  config                  0x400000000
  sample_type             IDENTIFIER
  read_format             TOTAL_TIME_ENABLED|TOTAL_TIME_RUNNING
  disabled                1
  inherit                 1
  enable_on_exec          1
  exclude_guest           1
.ft
```

and

```
.ft C
perf_event_attr:
  size                    120
  config                  0x800000000
  sample_type             IDENTIFIER
  read_format             TOTAL_TIME_ENABLED|TOTAL_TIME_RUNNING
  disabled                1
  inherit                 1
  enable_on_exec          1
```

exclude_guest 1

.ft

Performance counter stats for 'taskset -c 16 ./triad_loop':

233,066,666 cpu_core/cycles/ (0.43%)

604,097,080 cpu_atom/cycles/ (99.57%)

perf-record:

If there is no -e specified in perf record, on hybrid platform, it creates two default cycles and adds them to event list. One is for core, the other is for atom.

perf-stat:

If there is no -e specified in perf stat, on hybrid platform, besides of software events, following events are created and added to event list in order.

cpu_core/cycles/, cpu_atom/cycles/, cpu_core/instructions/, cpu_atom/instructions/, cpu_core/branches/, cpu_atom/branches/, cpu_core/branch-misses/, cpu_atom/branch-misses/

Of course, both perf-stat and perf-record support to enable hybrid event with a specific pmu.

e.g. perf stat -e cpu_core/cycles/ perf stat -e cpu_atom/cycles/ perf stat -e cpu_core/r1a/ perf stat -e cpu_atom/L1-icache-loads/ perf stat -e cpu_core/cycles/,cpu_atom/instructions/ perf stat -e {cpu_core/cycles/,cpu_core/instructions/}

But {cpu_core/cycles/,cpu_atom/instructions/} will return warning and disable grouping, because the pmus in group are not matched (cpu_core vs. cpu_atom).

JSON FORMAT

With -j, perf stat is able to print out a JSON format output that can be used for parsing.

? interval : optional timestamp in fractions of second (with -I)

? optional aggregate options:

? core : core identifier (with --per-core)

? die : die identifier (with --per-die)

? socket : socket identifier (with --per-socket)

? node : node identifier (with --per-node)

? thread : thread identifier (with --per-thread)

? counters : number of aggregated PMU counters

? counter-value : counter value

? unit : unit of the counter value or empty

? event : event name

- ? variance : optional variance if multiple values are collected (with -r)
- ? event-runtime : run time of the event
- ? pcnt-running : percentage of time the event was running
- ? metric-value : optional metric value
- ? metric-unit : optional unit of metric

SEE ALSO

perf-top(1), perf-list(1)

perf 05/25/2026 PERF-STAT(1)