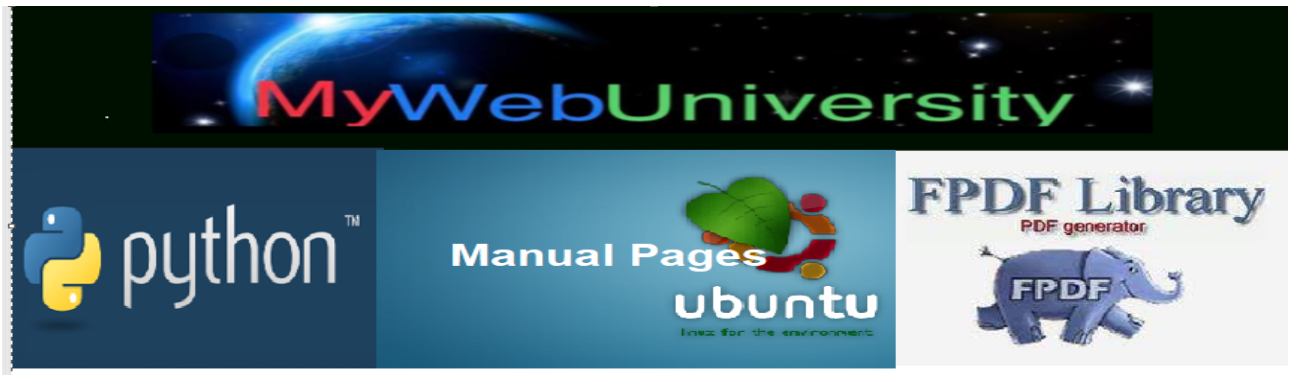

Full credit is given to the above companies including the OS that this PDF file was generated!

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

$ man perf-list.1

PERF-LIST(1) perf Manual PERF-LIST(1)

NAME

perf-list - List all symbolic event types

SYNOPSIS

perf list [<options>]

[hw|sw|cache|tracepoint|pmu|sdt|metric|metricgroup|event_glob]

DESCRIPTION

This command displays the symbolic event types which can be selected in the various perf commands with the -e option.

OPTIONS

-d, --desc

Print extra event descriptions. (default)

--no-desc

Don?t print descriptions.

-v, --long-desc

Print longer event descriptions and all similar PMUs with alphanumeric suffixes.

--debug

Enable debugging output.

--details

Print how named events are resolved internally into perf events, and also any extra expressions computed by perf stat.

--deprecated

Print deprecated events. By default the deprecated events are hidden.

--unit

Print PMU events and metrics limited to the specific PMU name. (e.g. --unit cpu, --unit msr, --unit cpu_core, --unit cpu_atom)

-j, --json

Output in JSON format.

-o, --output=

Output file name. By default output is written to stdout.

EVENT MODIFIERS

Events can optionally have a modifier by appending a colon and one or more modifiers. Modifiers allow the user to restrict the events to be counted. The following modifiers exist:

u - user-space counting

k - kernel counting

h - hypervisor counting

I - non idle counting

G - guest counting (in KVM guests)

H - host counting (not in KVM guests)

p - precise level

P - use maximum detected precise level

S - read sample value (PERF_SAMPLE_READ)

D - pin the event to the PMU

W - group is weak and will fallback to non-group if not schedulable,

e - group or event are exclusive and do not share the PMU

b - use BPF aggregration (see perf stat --bpf-counters)

R - retire latency value of the event

X - don't regroup the event to match PMUs

The p modifier can be used for specifying how precise the instruction address should be. The p modifier can be specified multiple times:

0 - SAMPLE_IP can have arbitrary skid

1 - SAMPLE_IP must have constant skid

2 - SAMPLE_IP requested to have 0 skid

3 - SAMPLE_IP must have 0 skid, or uses randomization to avoid sample shadowing effects.

For Intel systems precise event sampling is implemented with PEBS which supports up to precise-level 2, and precise level 3 for some special cases

On AMD systems it is implemented using IBS OP (up to precise-level 2). Unlike Intel PEBS which provides levels of precision, AMD core pmu is inherently non-precise and IBS is inherently precise. (i.e. ibs_op//, ibs_op//p, ibs_op//pp and ibs_op//ppp are all same). The precise modifier works with event types 0x76 (cpu-cycles, CPU clocks not halted) and 0xC1 (micro-ops retired). Both events map to IBS execution sampling (IBS op) with the IBS Op Counter Control bit (IbsOpCntCtl) set respectively (see the Core Complex (CCX) ? Processor x86 Core ? Instruction Based Sampling (IBS) section of the [AMD Processor Programming Reference (PPR)] relevant to the family, model and stepping of the processor being used).

Manual Volume 2: System Programming, 13.3 Instruction-Based Sampling). Examples to use IBS:

```
perf record -a -e cpu-cycles:p ...    # use ibs op counting cycles
perf record -a -e r076:p ...          # same as -e cpu-cycles:p
perf record -a -e r0C1:p ...          # use ibs op counting micro-ops
```

RAW HARDWARE EVENT DESCRIPTOR

Even when an event is not available in a symbolic form within perf right now, it can be encoded in a per processor specific way.

For instance on x86 CPUs, N is a hexadecimal value that represents the raw register encoding with the layout of IA32_PERFEVTSELx MSRs (see [Intel? 64 and IA-32 Architectures Software Developer?s Manual Volume 3B: System Programming Guide] Figure 30-1 Layout of IA32_PERFEVTSELx MSRs) or AMD?s PERF_CTL MSRs (see the Core Complex (CCX) ? Processor x86 Core ? MSR Registers section of the [AMD Processor Programming Reference (PPR)] relevant to the family, model and stepping of the processor being used).

Note: Only the following bit fields can be set in x86 counter registers: event, umask, edge, inv, cmask. Esp. guest/host only and OS/user mode flags must be setup using EVENT MODIFIERS.

Example:

If the Intel docs for a QM720 Core i7 describe an event as:

```
    Event  Umask  Event Mask
    Num.   Value  Mnemonic    Description                    Comment
    A8H     01H  LSD.UOPS    Counts the number of micro-ops     Use cmask=1 and
                             delivered by loop stream detector  invert to count
                                                                cycles
```

raw encoding of 0x1A8 can be used:

perf stat -e r1a8 -a sleep 1

perf record -e r1a8 ...

It?s also possible to use pmu syntax:

perf record -e r1a8 -a sleep 1

perf record -e cpu/r1a8/ ...

perf record -e cpu/r0x1a8/ ...

Some processors, like those from AMD, support event codes and unit masks larger than a byte.

In such cases, the bits corresponding to the event configuration parameters can be seen

with:

cat /sys/bus/event_source/devices/<pmu>/format/<config>

Example:

If the AMD docs for an EPYC 7713 processor describe an event as:

Event Umask Event Mask

Num. Value Mnemonic Description

28FH 03H op_cache_hit_miss.op_cache_hit Counts Op Cache micro-tag

hit events.

raw encoding of 0x0328F cannot be used since the upper nibble of the EventSelect bits have

to be specified via bits 32-35 as can be seen with:

cat /sys/bus/event_source/devices/cpu/format/event

raw encoding of 0x20000038F should be used instead:

perf stat -e r20000038f -a sleep 1

perf record -e r20000038f ...

It?s also possible to use pmu syntax:

perf record -e r20000038f -a sleep 1

perf record -e cpu/r20000038f/ ...

perf record -e cpu/r0x20000038f/ ...

You should refer to the processor specific documentation for getting these details. Some of

them are referenced in the SEE ALSO section below.

ARBITRARY PMUS

perf also supports an extended syntax for specifying raw parameters to PMUs. Using this

typically requires looking up the specific event in the CPU vendor specific documentation.

The available PMUs and their raw parameters can be listed with

ls /sys/bus/event_source/devices/*/format

For example the raw event "LSD.UOPS" core pmu event above could be specified as

perf stat -e cpu/event=0xa8,umask=0x1,name=LSD.UOPS_CYCLES,cmask=0x1/ ...

or using extended name syntax

perf stat -e cpu/event=0xa8,umask=0x1,cmask=0x1,name=\'LSD.UOPS_CYCLES:cmask=0x1\'/ ...

PER SOCKET PMUS

Some PMUs are not associated with a core, but with a whole CPU socket. Events on these PMUs generally cannot be sampled, but only counted globally with perf stat -a. They can be bound to one logical CPU, but will measure all the CPUs in the same socket.

This example measures memory bandwidth every second on the first memory controller on socket 0 of a Intel Xeon system

perf stat -C 0 -a uncore_imc_0/cas_count_read/,uncore_imc_0/cas_count_write/ -I 1000 ...

Each memory controller has its own PMU. Measuring the complete system bandwidth would require specifying all imc PMUs (see perf list output), and adding the values together. To simplify creation of multiple events, prefix and glob matching is supported in the PMU name, and the prefix uncore_ is also ignored when performing the match. So the command above can be expanded to all memory controllers by using the syntaxes:

perf stat -C 0 -a imc/cas_count_read/,imc/cas_count_write/ -I 1000 ...

perf stat -C 0 -a *imc*/cas_count_read/,*imc*/cas_count_write/ -I 1000 ...

This example measures the combined core power every second

perf stat -I 1000 -e power/energy-cores/ -a

ACCESS RESTRICTIONS

For non root users generally only context switched PMU events are available. This is normally only the events in the cpu PMU, the predefined events like cycles and instructions and some software events.

Other PMUs and global measurements are normally root only. Some event qualifiers, such as "any", are also root only.

This can be overridden by setting the kernel.perf_event_paranoid sysctl to -1, which allows non root to use these events.

For accessing trace point events perf needs to have read access to /sys/kernel/tracing, even when perf_event_paranoid is in a relaxed setting.

TOOL/HWMON EVENTS

Some events don?t have an associated PMU instead reading values available to software without perf_event_open. As these events don?t support sampling they can only really be read

by tools like perf stat.

Tool events provide times and certain system parameters. Examples include duration_time, user_time, system_time and num_cpus_online.

Hwmon events provide easy access to hwmon sysfs data typically in /sys/class/hwmon. This information includes temperatures, fan speeds and energy usage.

TRACING

Some PMUs control advanced hardware tracing capabilities, such as Intel PT, that allows low overhead execution tracing. These are described in a separate intel-pt.txt document.

PARAMETERIZED EVENTS

Some pmu events listed by perf-list will be displayed with ? in them. For example:

```
hv_gpci/dtbp_ptitc,phys_processor_idx=?/
```

This means that when provided as an event, a value for ? must also be supplied. For example:

```
perf stat -C 0 -e 'hv_gpci/dtbp_ptitc,phys_processor_idx=0x2/' ...
```

EVENT QUALIFIERS

It is also possible to add extra qualifiers to an event:

percore:

Sums up the event counts for all hardware threads in a core, e.g.:

```
perf stat -e cpu/event=0,umask=0x3,percore=1/
```

cpu:

Specifies a CPU or a range of CPUs to open the event upon. It may also reference a PMU to copy the CPU mask from. The value may be repeated to specify opening the event on multiple CPUs.

Example 1: to open the instructions event on CPUs 0 and 2, the cycles event on CPUs 1 and 2:

```
perf stat -e instructions/cpu=0,cpu=2/,cycles/cpu=1-2/ -a sleep 1
```

Example 2: to open the data_read uncore event on CPU 0 and the data_write uncore event on CPU 1:

```
perf stat -e data_read/cpu=0/,data_write/cpu=1/ -a sleep 1
```

Example 3: to open the software msr/tsc/ event only on the CPUs matching those from the cpu_core PMU:

```
perf stat -e msr/tsc,cpu=cpu_core/ -a sleep 1
```

EVENT GROUPS

Perf supports time based multiplexing of events, when the number of events active exceeds

the number of hardware performance counters. Multiplexing can cause measurement errors when the workload changes its execution profile.

When metrics are computed using formulas from event counts, it is useful to ensure some events are always measured together as a group to minimize multiplexing errors. Event groups can be specified using { }.

```
perf stat -e '{instructions,cycles}' ...
```

The number of available performance counters depend on the CPU. A group cannot contain more events than available counters. For example Intel Core CPUs typically have four generic performance counters for the core, plus three fixed counters for instructions, cycles and ref-cycles. Some special events have restrictions on which counter they can schedule, and may not support multiple instances in a single group. When too many events are specified in the group some of them will not be measured.

Globally pinned events can limit the number of counters available for other groups. On x86 systems, the NMI watchdog pins a counter by default. The nmi watchdog can be disabled as root with

```
echo 0 > /proc/sys/kernel/nmi_watchdog
```

Events from multiple different PMUs cannot be mixed in a group, with some exceptions for software events.

LEADER SAMPLING

perf also supports group leader sampling using the :S specifier.

```
perf record -e '{cycles,instructions}:S' ...
perf report --group
```

Normally all events in an event group sample, but with :S only the first event (the leader) samples, and it only reads the values of the other events in the group.

However, in the case AUX area events (e.g. Intel PT or CoreSight), the AUX area event must be the leader, so then the second event samples, not the first.

OPTIONS

Without options all known events will be listed.

To limit the list use:

1. hw or hardware to list hardware events such as cache-misses, etc.
2. sw or software to list software events such as context switches, etc.
3. cache or hwcache to list hardware cache events such as L1-dcache-loads, etc.
4. tracepoint to list all tracepoint events, alternatively use subsys_glob:event_glob to

filter by tracepoint subsystems such as sched, block, etc.

5. pmu to print the kernel supplied PMU events.
6. sdt to list all Statically Defined Tracepoint events.
7. metric to list metrics
8. metricgroup to list metricgroups with metrics.
9. If none of the above is matched, it will apply the supplied glob to all events, printing
   the ones that match.
10. As a last resort, it will do a substring search in all event names.

One or more types can be used at the same time, listing the events for the types specified.

Support raw format:

1. --raw-dump, shows the raw-dump of all the events.
2. --raw-dump [hw|sw|cache|tracepoint|pmu|event_glob], shows the raw-dump of a certain kind
   of events.

INTEL AUTO COUNTER RELOAD SUPPORT

Support for Intel Auto Counter Reload in perf tools

Auto counter reload provides a means for software to specify to hardware that certain counters, if supported, should be automatically reloaded upon overflow of chosen counters. By taking a sample only if the rate of one event exceeds some threshold relative to the rate of another event, this feature enables software to sample based on the relative rate of two or more events. To enable this, the user must provide a sample period term and a bitmask ("acr_mask") for each relevant event specifying the counters in an event group to reload if the event?s specified sample period is exceeded.

For example, if the user desires to measure a scenario when IPC > 2, the event group might look like the one below:

```
perf record -e {cpu_atom/instructions,period=200000,acr_mask=0x2/, \
cpu_atom/cycles,period=100000,acr_mask=0x3/} -- true
```

In this case, if the "instructions" counter exceeds the sample period of 200000, the second counter, "cycles", will be reset and a sample will be taken. If "cycles" is exceeded first, both counters in the group will be reset. In this way, samples will only be taken for cases where IPC > 2.

The acr_mask term is a hexadecimal value representing a bitmask of the events in the group to be reset when the period is exceeded. In the example above, "instructions" is assigned an acr_mask of 0x2, meaning only the second event in the group is reloaded and a sample is

taken for the first event. "cycles" is assigned an acr_mask of 0x3, meaning that both event counters will be reset if the sample period is exceeded first.

RATIO-TO-PREV EVENT TERM

To simplify this, an event term "ratio-to-prev" is provided which is used alongside the sample period term n or the -c/--count option. This would allow users to specify the desired relative rate between events as a ratio. Note: Both events compared must belong to the same PMU.

The command above would then become

```
perf record -e {cpu_atom/instructions/, \
cpu_atom/cycles,period=100000,ratio-to-prev=0.5/} -- true
```

ratio-to-prev is the ratio of the event using the term relative to the previous event in the group, which will always be 1, for a 1:0.5 or 2:1 ratio.

To sample for IPC < 2 for example, the events need to be reordered:

```
perf record -e {cpu_atom/cycles/, \
cpu_atom/instructions,period=200000,ratio-to-prev=2.0/} -- true
```

SEE ALSO

perf-stat(1), perf-top(1), perf-record(1), Intel? 64 and IA-32 Architectures Software Developer?s Manual Volume 3B: System Programming Guide[1], AMD Processor Programming Reference (PPR)[2]

NOTES

1. Intel? 64 and IA-32 Architectures Software Developer?s Manual Volume 3B: System Programming Guide
   http://www.intel.com/sdm/
2. AMD Processor Programming Reference (PPR)
   https://bugzilla.kernel.org/show_bug.cgi?id=206537