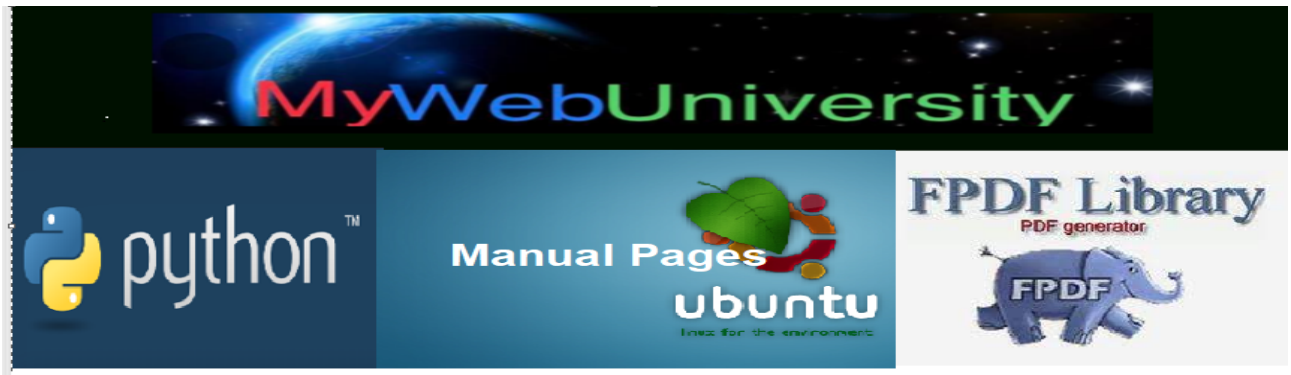

Full credit is given to the above companies including the OS that this PDF file was generated!

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

$ man perf-arm-spe.1

PERF-ARM-SPE(1) perf Manual PERF-ARM-SPE(1)

NAME

perf-arm-spe - Support for Arm Statistical Profiling Extension within Perf tools

SYNOPSIS

perf record -e arm_spe//

DESCRIPTION

The SPE (Statistical Profiling Extension) feature provides accurate attribution of latencies and events down to individual instructions. Rather than being interrupt-driven, it picks an instruction to sample and then captures data for it during execution. Data includes execution time in cycles. For loads and stores it also includes data address, cache miss events, and data origin.

The sampling has 5 stages:

1. Choose an operation
2. Collect data about the operation
3. Optionally discard the record based on a filter
4. Write the record to memory
5. Interrupt when the buffer is full

Choose an operation

This is chosen from a sample population, for SPE this is an IMPLEMENTATION DEFINED choice of all architectural instructions or all micro-ops. Sampling happens at a programmable interval. The architecture provides a mechanism for the SPE driver to infer the minimum interval at which it should sample. This minimum interval is used by the driver if no interval is specified. A pseudo-random perturbation is also added to the sampling interval

by default.

Collect data about the operation

Program counter, PMU events, timings and data addresses related to the operation are recorded. Sampling ensures there is only one sampled operation is in flight.

Optionally discard the record based on a filter

Based on programmable criteria, choose whether to keep the record or discard it. If the record is discarded then the flow stops here for this sample.

Write the record to memory

The record is appended to a memory buffer

Interrupt when the buffer is full

When the buffer fills, an interrupt is sent and the driver signals Perf to collect the records. Perf saves the raw data in the perf.data file.

OPENING THE FILE

Up until this point no decoding of the SPE data was done by either the kernel or Perf. Only when the recorded file is opened with perf report or perf script does the decoding happen. When decoding the data, Perf generates "synthetic samples" as if these were generated at the time of the recording. These samples are the same as if normal sampling was done by Perf without using SPE, although they may have more attributes associated with them. For example a normal sample may have just the instruction pointer, but an SPE sample can have data addresses and latency attributes.

WHY SAMPLING?

- ? Sampling, rather than tracing, cuts down the profiling problem to something more manageable for hardware. Only one sampled operation is in flight at a time.
- ? Allows precise attribution data, including: Full PC of instruction, data virtual and physical addresses.
- ? Allows correlation between an instruction and events, such as TLB and cache miss. (Data source indicates which particular cache was hit, but the meaning is implementation defined because different implementations can have different cache configurations.)

However, SPE does not provide any call-graph information, and relies on statistical methods.

COLLISIONS

When an operation is sampled while a previous sampled operation has not finished, a collision occurs. The new sample is dropped. Collisions affect the integrity of the data, so the sample rate should be set to avoid collisions.

The sample_collision PMU event can be used to determine the number of lost samples. Although this count is based on collisions before filtering occurs. Therefore this can not be used as an exact number for samples dropped that would have made it through the filter, but can be a rough guide.

THE EFFECT OF MICROARCHITECTURAL SAMPLING

If an implementation samples micro-operations instead of instructions, the results of sampling must be weighted accordingly.

For example, if a given instruction A is always converted into two micro-operations, A0 and A1, it becomes twice as likely to appear in the sample population.

The coarse effect of conversions, and, if applicable, sampling of speculative operations, can be estimated from the sample_pop and inst_retired PMU events.

KERNEL REQUIREMENTS

The ARM_SPE_PMU config must be set to build as either a module or statically.

Depending on CPU model, the kernel may need to be booted with page table isolation disabled (kpti=off). If KPTI needs to be disabled, this will fail with a console message "profiling buffer inaccessible. Try passing kpti=off on the kernel command line".

For the full criteria that determine whether KPTI needs to be forced off or not, see function unmap_kernel_at_el0() in the kernel sources. Common cases where it?s not required are on the CPUs in kpti_safe_list, or on Arm v8.5+ where FEAT_E0PD is mandatory.

The SPE interrupt must also be described by the firmware. If the module is loaded and KPTI is disabled (or isn?t required to be disabled) but the SPE PMU still doesn?t show in /sys/bus/event_source/devices/, then it?s possible that the SPE interrupt isn?t described by ACPI or DT. In this case no warning will be printed by the driver.

CAPTURING SPE WITH PERF COMMAND-LINE TOOLS

You can record a session with SPE samples:

```
perf record -e arm_spe// -- ./mybench
```

The sample period is set from the -c option, and because the minimum interval is used by default it?s recommended to set this to a higher value. The value is written to PMSIRR.INTERVAL.

Config parameters

These are placed between the // in the event and comma separated. For example -e arm_spe/load_filter=1,min_latency=10/

event_filter=<mask> - logical AND filter on specific events (PMSEVFR) - see bitfield description below

inv_event_filter=<mask> - logical OR to filter out specific events (PMSNEVFR, FEAT_SPEv1p2) - see bitfield description below

jitter=1 - use jitter to avoid resonance when sampling (PMSIRR.RND)

min_latency=<n> - collect only samples with this latency or higher* (PMSLATFR)

pa_enable=1 - collect physical address (as well as VA) of loads/stores (PMSCR.PA) - requires privilege

pct_enable=1 - collect physical timestamp instead of virtual timestamp (PMSCR.PCT) - requires privilege

ts_enable=1 - enable timestamping with value of generic timer (PMSCR.TS)

discard=1 - enable SPE PMU events but don't collect sample data - see 'Discard mode' (PMBLIMITR.FM = DISCARD)

inv_data_src_filter=<mask> - mask to filter from 0-63 possible data sources (PMSDSFR, FEAT_SPE_FDS) - See 'Data source filtering'

* Latency is the total latency from the point at which sampling started on that instruction,
rather than only the execution latency.

Only some events can be filtered on using event_filter bits. The overall filter is the
logical AND of these bits, for example if bits 3 and 5 are set only samples that have both
L1D cache refill AND TLB walk are recorded. When FEAT_SPEv1p2 is implemented
inv_event_filter can also be used to exclude events that have any (OR) of the filter?s bits
set. For example setting bits 3 and 5 in inv_event_filter will exclude any events that are
either L1D cache refill OR TLB walk. If the same bit is set in both filters it?s
UNPREDICTABLE whether the sample is included or excluded. Filter bits for both event_filter
and inv_event_filter are:

bit 1 - Instruction retired (i.e. omit speculative instructions)

bit 2 - L1D access (FEAT_SPEv1p4)

bit 3 - L1D refill

bit 4 - TLB access (FEAT_SPEv1p4)

bit 5 - TLB refill

bit 6 - Not taken event (FEAT_SPEv1p2)

bit 7 - Mispredict

bit 8 - Last level cache access (FEAT_SPEv1p4)

bit 9 - Last level cache miss (FEAT_SPEv1p4)

bit 10 - Remote access (FEAT_SPEv1p4)

bit 11 - Misaligned access (FEAT_SPEv1p1)

bit 12-15 - IMPLEMENTATION DEFINED events (when implemented)

bit 17 - Partial or empty SME or SVE predicate (FEAT_SPEv1p1)

bit 18 - Empty SME or SVE predicate (FEAT_SPEv1p1)

bit 19 - L2D access (FEAT_SPEv1p4)

bit 20 - L2D miss (FEAT_SPEv1p4)

bit 21 - Cache data modified (FEAT_SPEv1p4)

bit 22 - Recently fetched (FEAT_SPEv1p4)

bit 23 - Data snooped (FEAT_SPEv1p4)

bit 24 - Streaming SVE mode event (when FEAT_SPE_SME is implemented), or IMPLEMENTATION DEFINED event 24 (when implemented, only versions less than FEAT_SPEv1p4)

bit 25 - SMCU or external coprocessor operation event when FEAT_SPE_SME is implemented, or IMPLEMENTATION DEFINED event 25 (when implemented, only versions less than FEAT_SPEv1p4)

bit 26-31 - IMPLEMENTATION DEFINED events (only versions less than FEAT_SPEv1p4)

bit 48-63 - IMPLEMENTATION DEFINED events (when implemented)

For IMPLEMENTATION DEFINED bits, refer to the CPU TRM if these bits are implemented.

The driver will reject events if requested filter bits require unimplemented SPE versions, but will not reject filter bits for unimplemented IMPDEF bits or when their related feature is not present (e.g. SME). For example, if FEAT_SPEv1p2 is not implemented, filtering on "Not taken event" (bit 6) will be rejected.

So to sample just retired instructions:

```
perf record -e arm_spe/event_filter=2/ -- ./mybench
```

or just mispredicted branches:

```
perf record -e arm_spe/event_filter=0x80/ -- ./mybench
```

When set, the following filters can be used to select samples that match any of the operation types (OR filtering). If only one is set then only samples of that type are collected:

branch_filter=1 - Collect branches (PMSFCR.B)

load_filter=1 - Collect loads (PMSFCR.LD)

store_filter=1 - Collect stores (PMSFCR.ST)

When extended filtering is supported (FEAT_SPE_EFT), SIMD and float pointer operations can also be selected:

simd_filter=1 - Collect SIMD loads, stores and operations (PMSFCR.SIMD)

float_filter=1 - Collect floating point loads, stores and operations (PMSFCR.FP)

When extended filtering is supported (FEAT_SPE_EFT), operation type filters can be changed to AND using _mask fields. For example samples could be selected if they are store AND SIMD by setting store_filter=1,simd_filter=1, store_filter_mask=1,simd_filter_mask=1. The new masks are as follows:

branch_filter_mask=1 - Change branch filter behavior from OR to AND (PMSFCR.Bm)

load_filter_mask=1 - Change load filter behavior from OR to AND (PMSFCR.LDm)

store_filter_mask=1 - Change store filter behavior from OR to AND (PMSFCR.STm)

simd_filter_mask=1 - Change SIMD filter behavior from OR to AND (PMSFCR.SIMDm)

float_filter_mask=1 - Change floating point filter behavior from OR to AND (PMSFCR.FPm)

Viewing the data

By default perf report and perf script will assign samples to separate groups depending on the attributes/events of the SPE record. Because instructions can have multiple events associated with them, the samples in these groups are not necessarily unique. For example perf report shows these groups:

Available samples

0 arm_spe//

0 dummy:u

21 l1d-miss

897 l1d-access

5 llc-miss

7 llc-access

2 tlb-miss

1K tlb-access

36 branch

0 remote-access

900 memory

1800 instructions

The arm_spe// and dummy:u events are implementation details and are expected to be empty. The instructions group contains the full list of unique samples that are not sorted into other groups. To generate only this group use --itrace=i1i.

1i (1 instruction interval) signifies no further downsampling. Rather than an instruction interval, this generates a sample every n SPE samples. For example to generate the default

set of events for every 100 SPE samples:

perf report --itrace==bxofmtMai100i

Other period types, for example nanoseconds (ns) are not currently supported.

Memory access details are also stored on the samples and this can be viewed with:

perf report --mem-mode

The latency value from the SPE sample is stored in the weight field of the Perf samples and can be displayed in Perf script and report outputs by enabling its display from the command line.

Common errors

? "Cannot find PMU ?arm_spe?. Missing kernel support?"

Module not built or loaded, KPTI not disabled, interrupt not described by firmware, or running on a VM. See 'Kernel Requirements' above.

? "Arm SPE CONTEXT packets not found in the traces."

Root privilege is required to collect context packets. But these only increase the accuracy of assigning PIDs to kernel samples. For userspace sampling this can be ignored.

? Excessively large perf.data file size

Increase sampling interval (see above)

PMU events

SPE has events that can be counted on core PMUs. These are prefixed with SAMPLE_, for example SAMPLE_POP, SAMPLE_FEED, SAMPLE_COLLISION and SAMPLE_FEED_BR. These events will only count when an SPE event is running on the same core that the PMU event is opened on, otherwise they read as 0. There are various ways to ensure that the PMU event and SPE event are scheduled together depending on the way the event is opened. For example opening both events as per-process events on the same process, although it?s not guaranteed that the PMU event is enabled first when context switching. For that reason it may be better to open the PMU event as a systemwide event and then open SPE on the process of interest.

Discard mode

SPE related (SAMPLE_* etc) core PMU events can be used without the overhead of collecting sample data if discard mode is supported (optional from Armv8.6). First run a system wide SPE session (or on the core of interest) using options to minimize output. Then run perf stat:

perf record -e arm_spe/discard/ -a -N -B --no-bpf-event -o - > /dev/null &

perf stat -e SAMPLE_FEED_LD

Data source filtering

When FEAT_SPE_FDS is present, inv_data_src_filter can be used as a mask to filter on a subset (0 - 63) of possible data source IDs. The full range of data sources is 0 - 65535 although these are unlikely to be used in practice. Data sources are IMPDEF so refer to the TRM for the mappings. Each bit N of the filter maps to data source N. The filter is an OR of all the bits, and the value provided inv_data_src_filter is inverted before writing to PMSDSFR_EL1 so that set bits exclude that data source and cleared bits include that data source. Therefore the default value of 0 is equivalent to no filtering (all data sources included).

For example, to include only data sources 0 and 3, clear bits 0 and 3 (0xFFFFFFFFFFFFFFF6)

When inv_data_src_filter is set to 0xFFFFFFFFFFFFFFFF, any samples with any data source set are excluded.

SEE ALSO

perf-record(1), perf-script(1), perf-report(1), perf-inject(1)