EMON User Guide

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

Contents

Notices and Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Revision History

Revision Number	Description	Revision Date
1.0	Initial release.	January 2013
2.0	Completed major documentation updates and renamed to EMON User’s Guide. Added Installation and Examples chapters. Updated examples in section 2.2 General Options: Collection.	February 2016
2.1	Added and removed event modifiers.	February 2017
2.2	Updated SEP driver version.	April 2017
2.3	Updated examples in chapters 3 and 4.	September 2017
2.4	Updated the guide with missing options and added description where required.	June 2018
2.5	Minor updates to commands.	February 2019
2.8	Added details about data collection on hybrid platforms	August 2023
2.9	Added -per-cpu-absolute-tsc option details	October 2023

1 About This Document

EMON is a low-level command-line tool that provides the ability to profile application and system performance. The tool leverages counters from hardware Performance Monitoring Units (PMUs) to collect performance monitoring events.

Users have the option of specifying hardware events and attributes. EMON allocates and configures the required event resources in the PMU to retrieve event counts from the processor core and uncore. The tool collects the number of occurrences of selected events for the duration of collection.

Intended Audience

This document is intended for developers who use EMON to analyze performance data.

Related Information

For information on Performance Monitoring Unit (PMU), go to http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html.

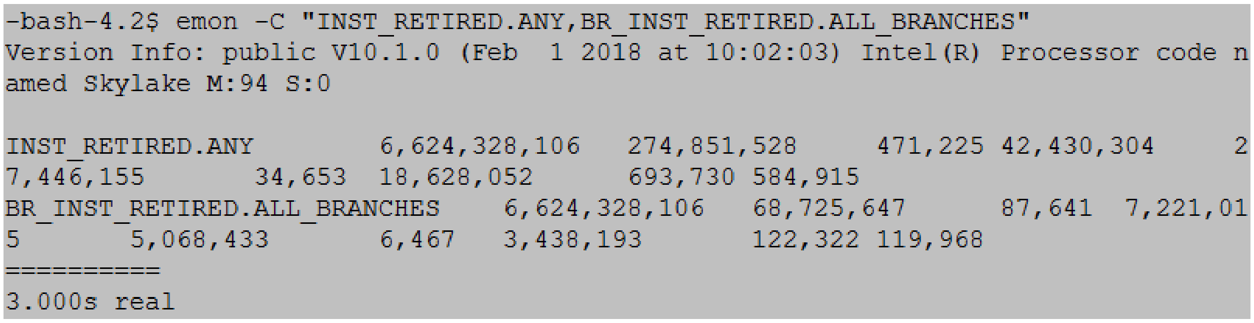
2 Usage

Use EMON with the following syntax:

```
emon [general-options] -C "event-definitions" [application-command-line]
```

The following example collects event data for `INST_RETIRED.ANY` and `BR_INST_RETIRED.ALL_BRANCHES` for a default duration of 3 s:

The output from EMON can be visualized as a table with multiple columns, as shown in the following image:



```
-bash-4.2$ emon -C "INST_RETIRED.ANY,BR_INST_RETIRED.ALL_BRANCHES"
Version Info: public V10.1.0 (Feb  1 2018 at 10:02:03) Intel(R) Processor code n
amed Skylake M:94 S:0

INST_RETIRED.ANY        6,624,328,106   274,851,528     471,225 42,430,304     2
7,446,155       34,653  18,628,052      693,730 584,915
BR_INST_RETIRED.ALL_BRANCHES    6,624,328,106   68,725,647      87,641  7,221,01
5       5,068,433       6,467   3,438,193       122,322 119,968
==========
3.000s real
```

The first column of EMON output contains the event name, and the second column contains clockticks spent during collection duration (6,624,328,106 in our example) followed by event counts on each processor core/ uncore unit. In this example, the platform contains eight logical cores. An event count column corresponds to each core.

Informative Options

This section lists all EMON options with examples to illustrate the behavior of certain options.

–h [-list-event-modifiers]

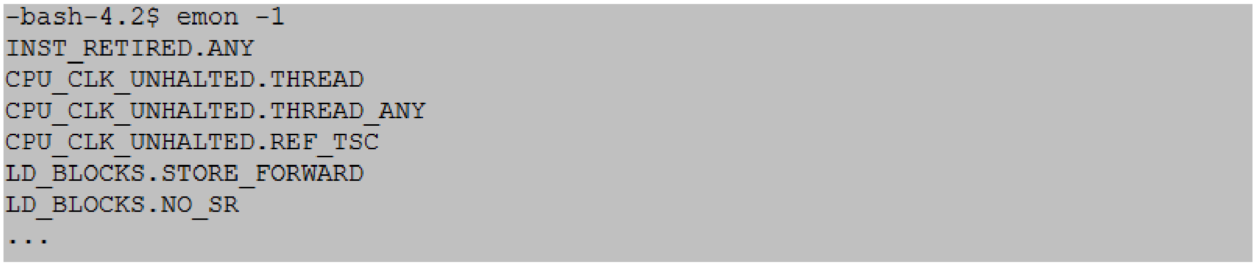
Display help information. The tool lists and describes all supported event modifiers if the sub-option `-list-event-modifiers` is specified. For details on event modifiers, see Event Modifiers.

–pmu-types [available]

Display the PMU types supported by the platform. Add the ‘available’ parameter to display PMU types available on the current system.

-1 [pmu-type] [-experimental | -all]

List the event names that can be monitored on the host platform. This command excludes events that are not available in the system even though the tool supports their collection. For example, if a system does not have an FPGA, all events related to FPGA are ignored.



```
-bash-4.2$ emon -1
INST_RETIRED.ANY
CPU_CLK_UNHALTED.THREAD
CPU_CLK_UNHALTED.THREAD_ANY
CPU_CLK_UNHALTED.REF_TSC
LD_BLOCKS.STORE_FORWARD
LD_BLOCKS.NO_SR
...
```

Event list can be filtered by adding a PMU type from `-pmu-types` command. For example:

```
emon -1 core
```

Experimental events are those events that have not been validated in hardware. To list experimental along with regular events, use the following command:

```
emon -1 -experimental
```

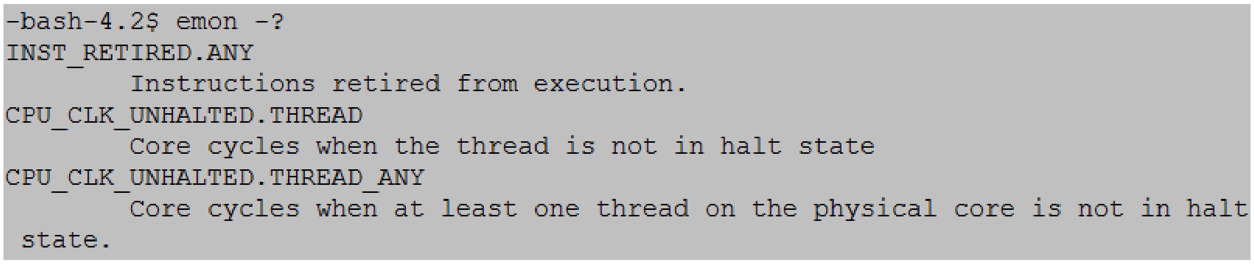
To list all events that the tool supports on the current platform, use the following command:

```
emon -1 -all
```

NOTE With `-all` option, the command lists experimental events, deprecated events, template events, and all other events enabled for the given platform.

-? | -H [pmu-type] [-experimental | -all]

Print events that can be monitored on the host platform along with a brief description. This command excludes events that are not available in the system even though the tool provides support for them. For example, if a system does not have an FPGA, all events related to FPGA are ignored.



```
-bash-4.2$ emon -?
INST_RETIRED.ANY
        Instructions retired from execution.
CPU_CLK_UNHALTED.THREAD
        Core cycles when the thread is not in halt state
CPU_CLK_UNHALTED.THREAD_ANY
        Core cycles when at least one thread on the physical core is not in halt
 state.
```

Event list can be filtered by adding a PMU type from `-pmu-types` command.

For example:

```
emon -? core
```

Experimental events are those events that have not been validated in hardware. To list experimental along with regular events, use the following command:

```
emon -? -experimental
```

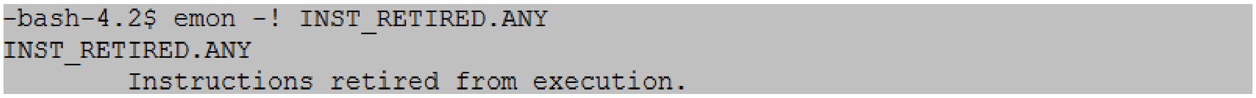
To list all events that the tool supports on the current platform, use the following command:

```
emon -? -all
```

NOTE This command lists experimental events, deprecated events, template events, and all other events enabled for the given platform.

-! <event name>

Print description of a given event.

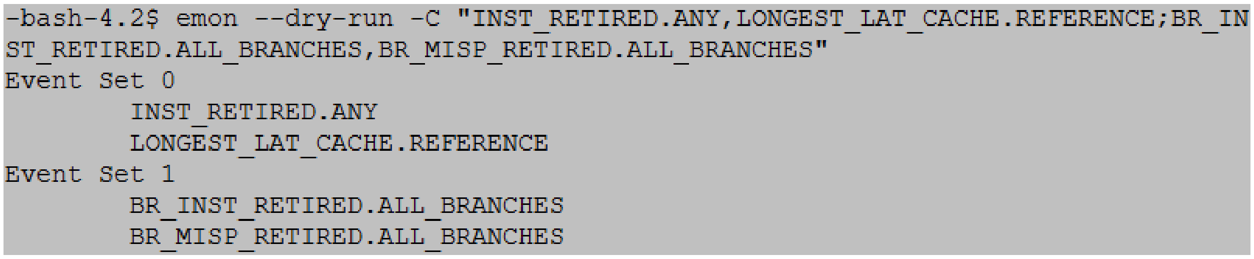


```
-bash-4.2$ emon -! INST_RETIRED.ANY
INST_RETIRED.ANY
        Instructions retired from execution.
```

If the given event does not have the relevant hardware support in the current system, EMON displays a warning saying that the event is not available for collection in the system.

--dry-run

Lists event groups with names of events that will be scheduled together. In the following example, EMON splits the command execution into two runs. The first execution includes events under `Event Set 0`, and second execution includes those under `Event Set 1`.



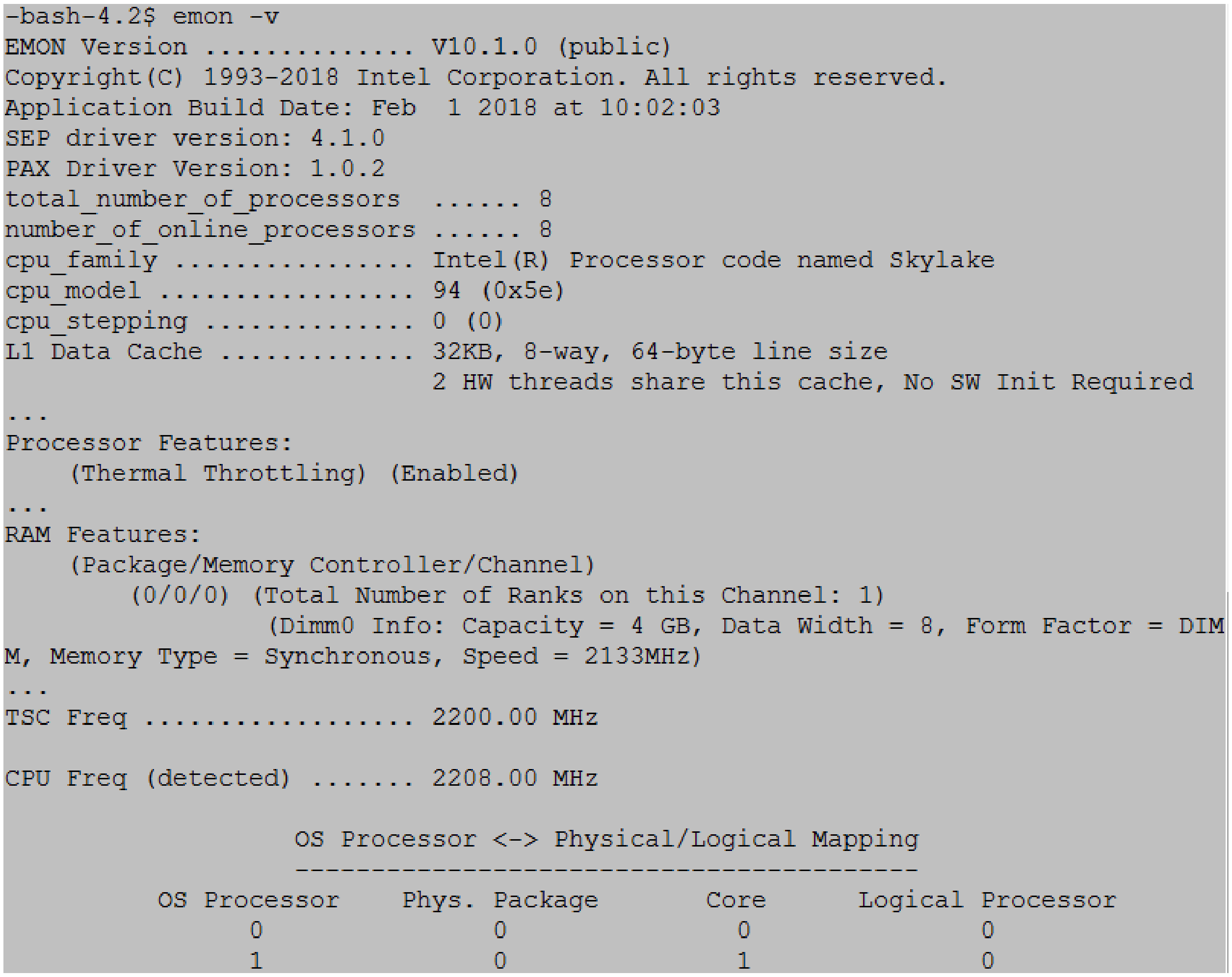
```
-bash-4.2$ emon --dry-run -C "INST_RETIRED.ANY,LONGEST_LAT_CACHE.REFERENCE;BR_IN
ST_RETIRED.ALL_BRANCHES,BR_MISP_RETIRED.ALL_BRANCHES"
Event Set 0
        INST_RETIRED.ANY
        LONGEST_LAT_CACHE.REFERENCE
Event Set 1
        BR_INST_RETIRED.ALL_BRANCHES
        BR_MISP_RETIRED.ALL_BRANCHES
```

-M

Print the operating system (OS) processor to hardware logical/physical processor mapping.

-v [-display-features]

Display build and version information of the tool along with other details about the hardware platform. This option also prints the mapping of the OS processor to the logical/physical processor of the hardware. Use the `-display-features` option with this command to display the capabilities supported in this version of driver.



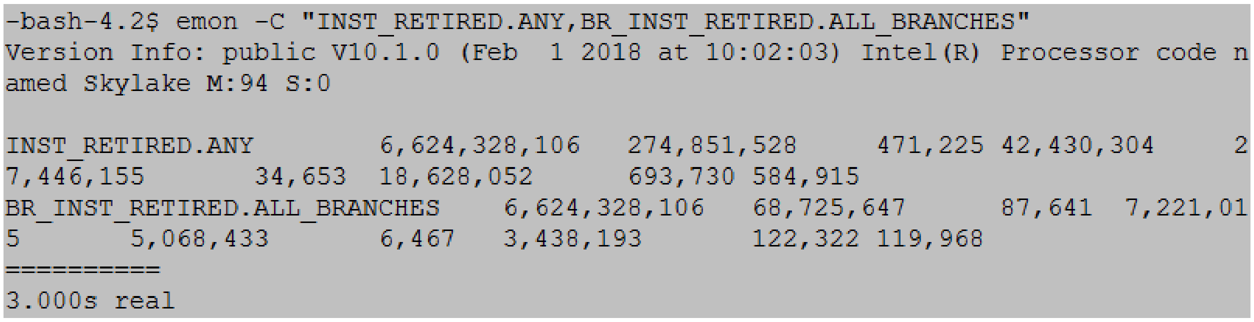
```
-bash-4.2$ emon -v
EMON Version .............. V10.1.0 (public)
Copyright(C) 1993-2018 Intel Corporation. All rights reserved.
Application Build Date: Feb  1 2018 at 10:02:03
SEP driver version: 4.1.0
PAX Driver Version: 1.0.2
total_number_of_processors  ...... 8
number_of_online_processors ...... 8
cpu_family ................ Intel(R) Processor code named Skylake
cpu_model ................. 94 (0x5e)
cpu_stepping .............. 0 (0)
L1 Data Cache ............. 32KB, 8-way, 64-byte line size
                            2 HW threads share this cache, No SW Init Required
...
Processor Features:
    (Thermal Throttling) (Enabled)
...
RAM Features:
    (Package/Memory Controller/Channel)
        (0/0/0) (Total Number of Ranks on this Channel: 1)
                 (Dimm0 Info: Capacity = 4 GB, Data Width = 8, Form Factor = DIM
M, Memory Type = Synchronous, Speed = 2133MHz)
...
TSC Freq .................. 2200.00 MHz

CPU Freq (detected) ....... 2208.00 MHz

                   OS Processor <-> Physical/Logical Mapping
                   -----------------------------------------
          OS Processor    Phys. Package       Core      Logical Processor
                0               0               0               0
                1               0               1               0
```

Event Collection Options

EMON collects event data for processor core and uncore. This section lists all EMON options related to data collection with examples to illustrate the behavior of certain options.

-C <event1,event2,...>

Specify one or more events for which the performance data will be collected. Events to monitor can optionally be embedded within double-quotes `(")` and should be separated by a comma `(,)`. Both core and uncore events can be specified for monitoring. However, when user specifies only uncore events in the command line, the tool collects all the fixed core events along with the specified uncore events.



```
-bash-4.2$ emon -C "INST_RETIRED.ANY,BR_INST_RETIRED.ALL_BRANCHES"
Version Info: public V10.1.0 (Feb  1 2018 at 10:02:03) Intel(R) Processor code n
amed Skylake M:94 S:0

INST_RETIRED.ANY        6,624,328,106   274,851,528     471,225 42,430,304     2
7,446,155       34,653  18,628,052      693,730 584,915
BR_INST_RETIRED.ALL_BRANCHES    6,624,328,106   68,725,647      87,641  7,221,01
5       5,068,433       6,467   3,438,193       122,322 119,968
==========
3.000s real
```

Data Collection and Event Multiplexing

The number of events that can be monitored simultaneously in a single run is limited by the number of hardware performance counters in the PMU of a processor. Certain events have restrictions that disallows their programming in all counters.

To overcome the limitation of available performance counters on the hardware, EMON splits events into multiple event groups. Each group consists of events that can be collected simultaneously in a single run. The tool schedules an independent data collection for each event group. Events are split in to multiple groups under following two conditions:

- If all events specified in the command line cannot fit into available performance counters on the platform, the tool automatically splits them in to multiple groups.
- User can control splitting of events in to groups while specifying event lists in the command line. To do so, use a semicolon to demarcate group separation instead of using a comma. To understand this use case, see Multi-group Core Events.

Event Modifiers

Individual core/uncore event behavior can be modified using modifiers. The `[:modifier=val]` option enables you to specify individual event modifiers along with the respective values for a given platform.

Event modifiers are attached to event names delimited by a colon `(:)`. They may or may not take values. Where applicable, values are of the following format: `<yes/no>`, `<0/1>`, `<dec/hex values>`. In some special cases explicitly mentioned, they could take other string values.

Basic Event Modifiers

The following table lists the basic event modifiers and provides a short description of each modifier.

Modifier	Description

`:USER |`
`:usr=<0/1>`
	Specifies that events are counted only when the processor is operating at privilege levels 1, 2, or 3. This flag can be used in conjunction with the SUP flag.

`:SUP|:os`	Specifies that events are counted only when the processor is operating at privilege level 0. This flag can be used in conjunction with the USER flag.
`:ALL`	Event data is collected regardless of the current privilege level.
`:cp`	In Check Point. When this modifier is specified, the data result will not include counts that occurred inside of an aborted Intel® Transactional Synchronization eXtensions (Intel® TSX) region.
`:tx`	In Transaction. When this modifier is specified, the data result will only include counts that occurred inside an Intel® TSX region, regardless of whether that region was aborted or committed.
`:perf_metrics`	Enable hardware based top-down metrics. This modifier is ignored on all events except for the fixed event `TOPDOWN.SLOTS`.
`:ocr_msr_val=<value>`	Override the default offcore MSR programming with the user specified value for the event.

Advanced Event Modifiers

The following table lists the event modifiers for more advanced users with an understanding of hardware PMU.

Modifier	Description
`:amt<0/1>`	Sets (1) or clears (0) the event's Any Thread control bit. A value of 0 causes the event to be counted on a per logical core basis, when applicable. A value of 1 causes the event to be counted on a per physical core basis. Please note that this feature is not supported on 10th generation Intel Core Processors and 3rd generation Intel Xeon Scalable Processors or newer.

`:c<cmask> |`
`:cmask=<cmask>`
	Value that will be compared to the count of the specified event during a single cycle per core. If the event count is greater than or equal to this value, the counter is incremented by one; otherwise, the counter is not incremented. The value must be in the range of 0x0 to 0xff.

`:e<0/1>`	Enables (when set) edge detection of the selected microarchitectural condition. The logical processor counts the number of deasserted to asserted transitions for any condition that can be expressed by the other fields.
	For example,
	`emon -l1 -t0.1 -C "MACHINE_CLEARS.COUNT,`
	`MACHINE_CLEARS.COUNT:e1:c1"`

`:i<0/1> |`
`:inv=<yes/no>`
	When the invert flag is set, inverts `cmask (:c<cmask> or :cmask=<cmask>)` comparison, so that both greater than or equal to and less than comparisons can be made (`<0>`: greater than equal to comparison, `<1>`: less than comparison).
	Invert flag is ignored when `:c<cmask>` is programmed to 0. A value of 0 disables invert and 1 enables it.

`:u<umask> |`
`:umask=<umask>`
	`<umask>` indicates the value of the event's unit mask to identify a specific microarchitectural condition. The `<umask>` value must be in the range 0x0 to 0xff.

`:p<0/1>`	When set, enables toggling of PMi pin for each event occurrence rather than during counter overflow.
`:request=<request name as string>`	Programming request type in the off-core response facility for a transaction request to the uncore. The request type specification must be accompanied by a response type.
`:response=<response name as string>`	Programming response type in the off-core response facility for a transaction request to the uncore. The response type specification must be accompanied by a request type.
`:t=<threshold_num>`	Threshold programming for uncore PMON_CTLx register. For events that increment more than 1 per cycle, if the threshold value is greater than 1, the data register will accumulate instances in which the event increment is >= threshold.

`:rx_match=<value>`
`:rx_mask=<value>`
`:tx_match=<value>`
`:tx_mask=<value>`
	Modifiers are all applicable to uncore Intel® QuickPath Interconnect (Intel® QPI) for programming filter registers.

Modifier	Description
:state=<value>	Applicable to uncore CHA to program state bit field of filter MSR_0.
:tid=<value>	Applicable to uncore CBO to program tid bit field of filter MSR_0.
:filter0=<value>	Applicable to CBO/CHA to program filter MSR_0.
:filter1=<value>	Applicable to CBO/CHA to program filter MSR_1.
:nc=<value>	Applicable to CBO/CHA to filter non-coherent requests by programming nc bit field of filter MSR_1.
:opc=<value>	Applicable to CBO/CHA to filter events based on their OPCODE by programming opc bit field of filter MSR_1.
:nid=<value>	Applicable to CBO/CHA to filter events by programming nid bit field of filter MSR_1.
:msr=<msr_index>	Read static and freerun event counts based on msr index provided in the command line.
:scope=<thread/Module/package>	Set scope for power events specified through :msr event modifier. The scope needs to be one of the 3 strings from the modifier column.
:type=<static/Freerun>	Set type of power events specified through :msr event modifier. The event type needs to be one of 2 string from the modifier column.
:ccst_debug=<hex_num>	Applicable to Power Control Unit (PCU) for programming debug MSR.
:umask_ext=<value>	Enables setting extended umask bits in the counter control register when used with applicable uncore events.

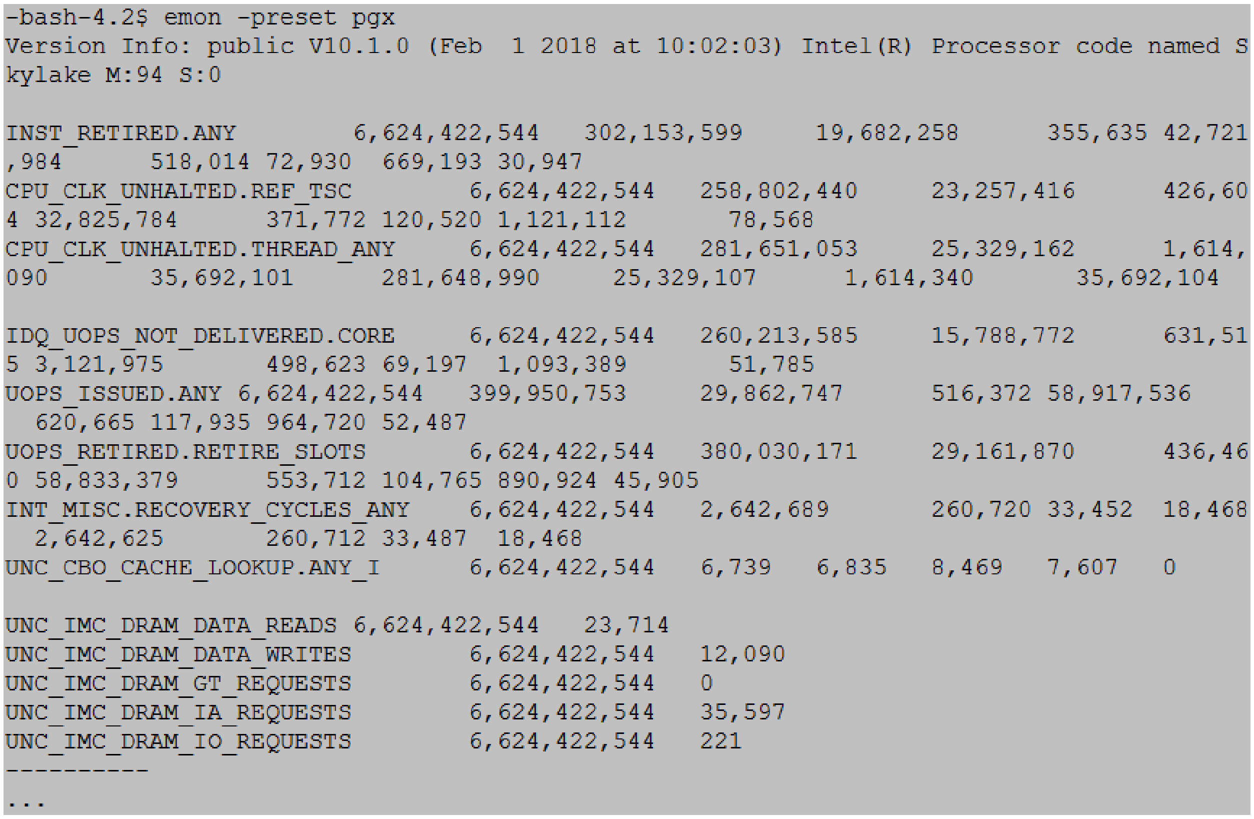
-preset-list

Presets are predefined event sets made available by the tool. This option lists all available presets.

```
-bash-4.2$ emon -preset-list
Preset[pgx] :
Platform Guided Exploration:  Top-Down analysis model for all system components

Preset[pgx] :

Preset[pgx] :
Platform Guided Exploration:  Top-Down analysis model for all system components
```

-preset <name>

Collect data for the given preset. To obtain available presets, use `emon -preset-list` command. Presets cannot be used along with `-C` option. When presets are used in combination with `-V` or `-S` options, EMON generates spreadsheet-friendly output.



```
-bash-4.2$ emon -preset pgx
Version Info: public V10.1.0 (Feb  1 2018 at 10:02:03) Intel(R) Processor code named S
kylake M:94 S:0

INST_RETIRED.ANY        6,624,422,544   302,153,599     19,682,258      355,635 42,721
,984      518,014 72,930  669,193 30,947
CPU_CLK_UNHALTED.REF_TSC        6,624,422,544   258,802,440     23,257,416      426,60
4 32,825,784      371,772 120,520 1,121,112       78,568
CPU_CLK_UNHALTED.THREAD_ANY     6,624,422,544   281,651,053     25,329,162      1,614,
090       35,692,101      281,648,990     25,329,107      1,614,340       35,692,104

IDQ_UOPS_NOT_DELIVERED.CORE     6,624,422,544   260,213,585     15,788,772      631,51
5 3,121,975       498,623 69,197  1,093,389       51,785
UOPS_ISSUED.ANY 6,624,422,544   399,950,753     29,862,747      516,372 58,917,536
  620,665 117,935 964,720 52,487
UOPS_RETIRED.RETIRE_SLOTS       6,624,422,544   380,030,171     29,161,870      436,46
0 58,833,379      553,712 104,765 890,924 45,905
INT_MISC.RECOVERY_CYCLES_ANY    6,624,422,544   2,642,689       260,720 33,452  18,468
  2,642,625       260,712 33,487  18,468
UNC_CBO_CACHE_LOOKUP.ANY_I      6,624,422,544   6,739   6,835   8,469   7,607   0

UNC_IMC_DRAM_DATA_READS 6,624,422,544   23,714
UNC_IMC_DRAM_DATA_WRITES        6,624,422,544   12,090
UNC_IMC_DRAM_GT_REQUESTS        6,624,422,544   0
UNC_IMC_DRAM_IA_REQUESTS        6,624,422,544   35,597
UNC_IMC_DRAM_IO_REQUESTS        6,624,422,544   221
----------
...
```

-t <time in sec>

Time (seconds) that an event set is monitored for. Default value is 3 s. To run EMON for the duration of application execution, use `-t0` along with an application. EMON kills the application after it finishes executing all given event sets for the specified duration when `-t0` is not specified.

The following command executes until matrix application finishes:

```
emon -t0 -C "INST_RETIRED.ANY" matrix "4 4096"
```

The following command kills the application and terminates after 10 s:

```
emon -t10 -C "INST_RETIRED.ANY" matrix "4 4096"
```

-l <loops>

The number of times each event set is monitored. Default value is 1. Event sets are interleaved.

For example, if two events sets A and B are specified and time equals 4 and loops equal 2, event set A is monitored for 4 seconds, and then event set B is monitored for 4 seconds, and then event set A is monitored for 4 seconds, and, finally, event set B is monitored for 4 seconds.

```
-bash-4.2$ emon -t4 -l2 -C "INST_RETIRED.ANY,BR_INST_RETIRED.ALL_BRANCHES"
Version Info: public V10.1.0 (Feb  1 2018 at 10:02:03) Intel(R) Processor code named S
kylake M:94 S:0

INST_RETIRED.ANY        8,832,408,718   402,554,887     319,429 1,557,350       56,881
,466    1,493   24,947,222      1,322,551       246,641
BR_INST_RETIRED.ALL_BRANCHES    8,832,408,718   98,360,811      52,483  287,100 9,662,
202     291     4,610,595       248,737 44,103
==========
INST_RETIRED.ANY        8,832,171,044   402,845,212     339,033 2,483,690       56,617
,358    1,493   25,033,773      921,139 83,362
BR_INST_RETIRED.ALL_BRANCHES    8,832,171,044   98,461,065      55,996  479,673 9,614,
240     291     4,627,796       161,736 15,905
==========
8.000s real
```

When launched with an application and the total monitoring time is less than application execution time, EMON kills the application after executing all loops. In the following example, each loop runs for 3 s for a total duration of 6 s, after which EMON would kill matrix application and exit:

```
emon -l2 -C "INST_RETIRED.ANY" matrix "16 8192"
```

When specified with an application and the total monitoring time is greater than application execution time, EMON continues executing loops in the remaining time. In the following example, each loop runs for 3 s for a total duration of 30 s while matrix application is expected to finish much sooner:

```
emon -l10 -C "INST_RETIRED.ANY" matrix "2 1024"
```

When specified with time 0 s and an application, EMON executes each loop for the duration of application execution. For example, in the following command assuming matrix application takes about 6 s to complete, each loop could run for ~6 s for a total duration of 18 s:

```
emon -t0 -l3 -C "INST_RETIRED.ANY" matrix "2 1024"
```

-L <time>

Range for random delay of the monitor interval, specified in seconds. A random delay of 0 s to `<time>` is introduced between each sample. When used, each monitor interval is the value of the `-t switch` plus the random delay between 0 and `<time>` milliseconds. Defaults to 0 m. This functionality will be automatically disabled if `-t switch` is set to 0 s.

```
-bash-4.2$ emon -t1 -L0.5 -l5 -C CPU_CLK_UNHALTED.REF_TSC
Version Info: public V10.1.0 (Feb  1 2018 at 10:02:03) Intel(R) Processor code named S
kylake M:94 S:0

CPU_CLK_UNHALTED.REF_TSC        2,208,846,728   85,338,832      151,524 478,400 10,972
,104    828,184 7,463,316       447,580 10,120
==========
CPU_CLK_UNHALTED.REF_TSC        2,208,170,252   85,321,996      129,076 1,024,236    1
0,923,988       64,492  7,502,600       847,596 5,888
==========
CPU_CLK_UNHALTED.REF_TSC        2,208,164,220   86,716,072      66,240  357,788 10,927
,392    55,568  7,531,028       373,796 5,336
==========
CPU_CLK_UNHALTED.REF_TSC        2,208,164,166   85,828,364      63,296  364,688 10,925
,736    56,304  7,487,052       353,188 5,796
==========
CPU_CLK_UNHALTED.REF_TSC        2,208,100,868   84,342,104      131,744 494,040 12,850
,928    3,681,840       10,954,900      3,626,180       9,789,352
==========
5.000s real
```

-s <delay>

One time delay in seconds before monitoring is started.

-w

Limit loops. The number of loops is limited by the application's execution time. For example, if the total monitoring time specified by the time and loop switches is greater than the actual application execution time, the collection is stopped after the application exits.

NOTE In the example below, even with `-l10`, EMON exits after first loop.

```
-bash-4.2$ emon -l10 -w -C "INST_RETIRED.ANY" matrix "1 256"
Version Info: public V10.1.0 (Feb  1 2018 at 10:02:03) Intel(R) Processor code named S
kylake M:94 S:0

Elapsed time = 0.000000 seconds
INST_RETIRED.ANY        6,624,654,314   283,997,362     24,737,446      47,771,456   6
,981,403        473,692,321     403,137 960,186 6,095,116
==========
3.000s real
```

-nb | -non-blocking

Start EMON collection in the background.

-p

Start EMON in paused state. If collection is never resumed, EMON exits after monitoring interval ends. In the following example, EMON would exit after 3 s if the collection is never resumed using `emon -resume`.

```
-bash-4.2$ emon -p -C "INST_RETIRED.ANY"
Version Info: public V10.1.0 (Feb  1 2018 at 10:02:03) Intel(R) Processor code named S
kylake M:94 S:0

Emon collector successfully paused.
Emon collector was started in PAUSE mode and never RESUMED
```

-osm | -os-mode

Collect data for operating system processes only.

-um | -user-mode

Collect data for user-mode processes only.

-pause

When EMON is running in non-blocking mode or in the background, use `emon -pause` to pause a running collection.

If EMON is running in the foreground, use the following steps to pause collection:

1. Open a Bash* shell, and then set up EMON run time environment by sourcing `sep_vars.sh` file in the current Bash* shell.

For example, if EMON is installed in `/opt/intel/emon, source /opt/intel/emon/sep_vars.sh`.

2. From the new shell, issue `emon -pause` to pause collection.

Collection ends if the total monitoring time elapses while paused.

-resume

When EMON is running in non-blocking mode or in the background, use `emon -resume` to resume a paused collection.

If EMON is running in the foreground, use the following steps to resume collection:

1. Open a Bash* shell, and then set up EMON run time environment by sourcing `sep_vars.sh` file in the current Bash* shell.

   For example, if EMON is installed in `/opt/intel/emon, source /opt/intel/emon/sep_vars.sh`.
2. From the new shell, issue `emon -resume` to resume collection.

-stop

When EMON is running in non-blocking mode or in the background, use `emon -stop` to stop a running collection.

If EMON is running in the foreground, use the following steps to stop collection:

1. Open a Bash* shell, and then set up EMON run time environment by sourcing `sep_vars.sh` file in the current Bash* shell.

   For example, if EMON is installed in `/opt/intel/emon source /opt/intel/emon/sep_vars.sh`.
2. From the new shell, issue `emon -stop` to stop collection.

Input/Output Options

This section lists all options related to tool input/output with examples to illustrate the behavior of certain options. The default output mode is text-based command-line output. Additionally, EMON provides options to generate text or spreadsheet output in to files.

-f <output file>

EMON output is written to `<output file>`. The `-f switch` creates a new output file.

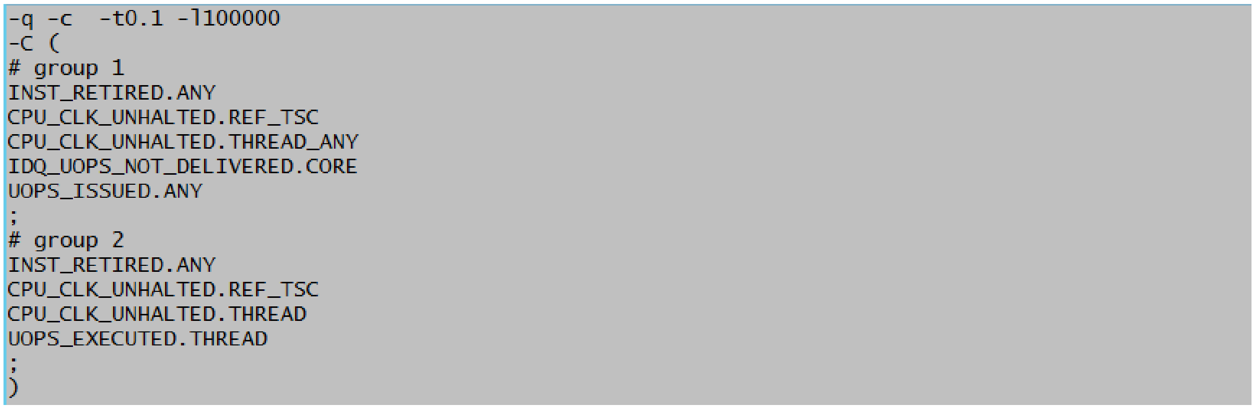
-F <output file>

EMON output is appended to `<output file>`. If `<output file>` does not exist, it will be created.

-i <input file>

EMON command-line arguments are provided by `<input file>`. Comments are indicated with a hashtag `(#)`. All text following a hashtag in an input file is ignored.

Create an input text file with desired options. Input options can be separated by spaces or new lines. Event list following `-C` can either use a new-line separator or a comma `(,)`. Use a semicolon `(;)` to start a new group.



```
-q -c  -t0.1 -l100000
-C (
# group 1
INST_RETIRED.ANY
CPU_CLK_UNHALTED.REF_TSC
CPU_CLK_UNHALTED.THREAD_ANY
IDQ_UOPS_NOT_DELIVERED.CORE
UOPS_ISSUED.ANY
;
# group 2
INST_RETIRED.ANY
CPU_CLK_UNHALTED.REF_TSC
CPU_CLK_UNHALTED.THREAD
UOPS_EXECUTED.THREAD
;
)
```

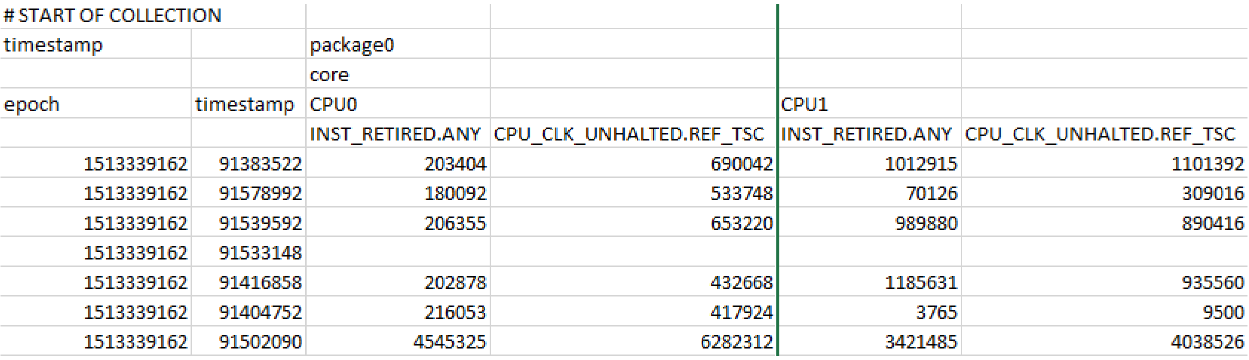
-q

Default text output to command line. Minimal information is output.

-V

EMON generates output in a spreadsheet-friendly format. Use `-f` or `-F` options to create spreadsheet-friendly output files.

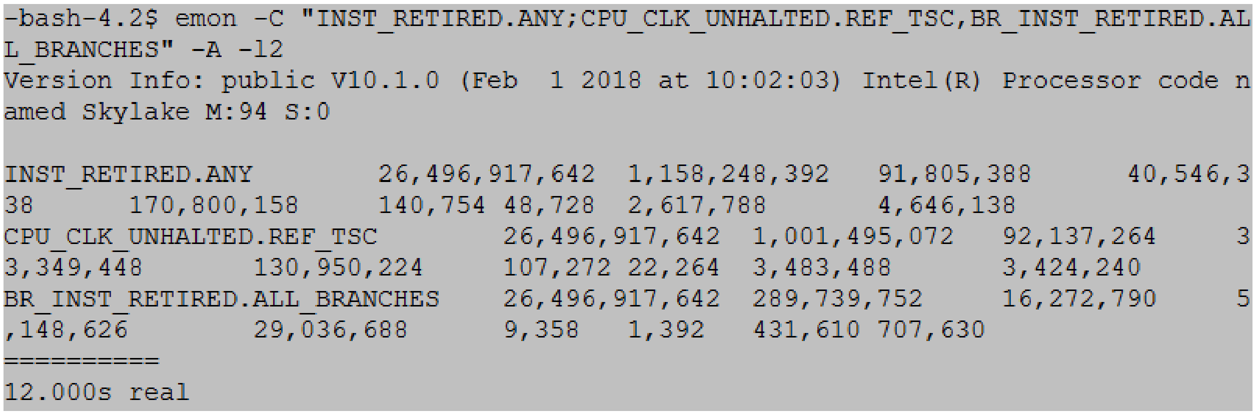
In this mode, data is hierarchically presented (`packages->devices->Specific Core/Uncore units->event counts`), making it easier to observe event counts on a particular core or uncore unit.



# START OF COLLECTION					
timestamp		package0			
		core			
epoch	timestamp	CPU0		CPU1	
		INST_RETIRED.ANY	CPU_CLK_UNHALTED.REF_TSC	INST_RETIRED.ANY	CPU_CLK_UNHALTED.REF_TSC
1513339162	91383522	203404	690042	1012915	1101392
1513339162	91578992	180092	533748	70126	309016
1513339162	91539592	206355	653220	989880	890416
1513339162	91533148				
1513339162	91416858	202878	432668	1185631	935560
1513339162	91404752	216053	417924	3765	9500
1513339162	91502090	4545325	6282312	3421485	4038526

-A

Display normalized event counts across all groups and loops in quiet mode output format.



```
-bash-4.2$ emon -C "INST_RETIRED.ANY;CPU_CLK_UNHALTED.REF_TSC,BR_INST_RETIRED.AL
L_BRANCHES" -A -l2
Version Info: public V10.1.0 (Feb  1 2018 at 10:02:03) Intel(R) Processor code n
amed Skylake M:94 S:0

INST_RETIRED.ANY         26,496,917,642  1,158,248,392   91,805,388      40,546,3
38       170,800,158     140,754 48,728  2,617,788       4,646,138
CPU_CLK_UNHALTED.REF_TSC         26,496,917,642  1,001,495,072   92,137,264      3
3,349,448        130,950,224     107,272 22,264  3,483,488       3,424,240
BR_INST_RETIRED.ALL_BRANCHES     26,496,917,642  289,739,752     16,272,790      5
,148,626         29,036,688      9,358   1,392   431,610 707,630
==========
12.000s real
```

To calculate the final counts:

1. Calculate normalized count for each event across groups (i.e., add counts of all occurrences of an event across groups and divide the accumulated value by actual number of occurrences of that event in the groups).
2. Multiply normalized count by total number of scheduled groups.
3. If there is more than one loop, repeat steps 1 and 2 for each loop and add corresponding event counts from each loop.

-S

Compute-tool defined performance metrics using normalized event counts and display in a semicolon-separated, spreadsheet-friendly format. The normalized event counts are calculated from raw event counts described in `-A` option. Use `-f` or `-F` options to create spreadsheet-friendly output files.

```
emon -preset pgx -S
```

-Sr

Behaves similar to `-S` option but additionally stores and displays raw event counts in a spreadsheet-friendly format.

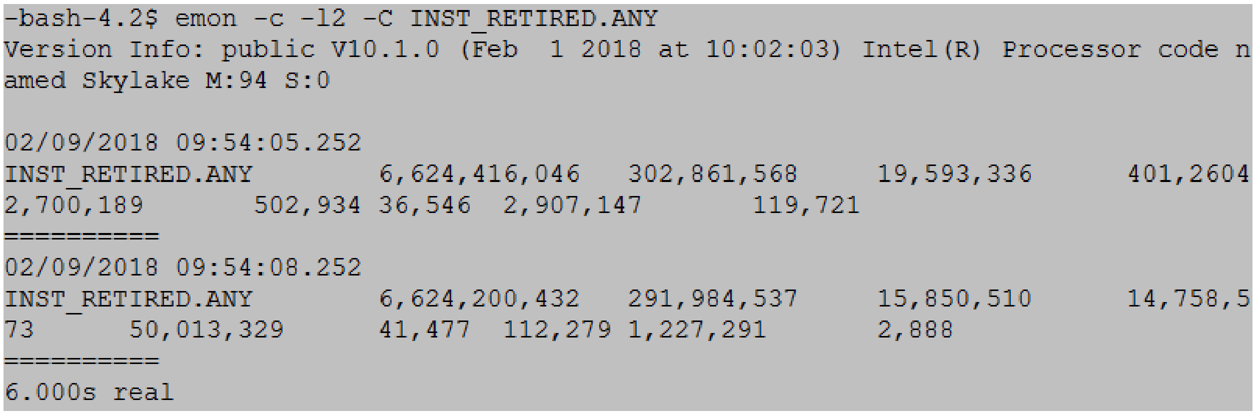
```
emon -preset pgx -Sr ./raw_counts_file.csv -f ./metrics_file.csv
```

-X

Spreadsheet-friendly format. The results are output in tab-separated format. This only works for single group collection.

-c

Print system time (date-time) for each time interval. It is only available in the command-line output.



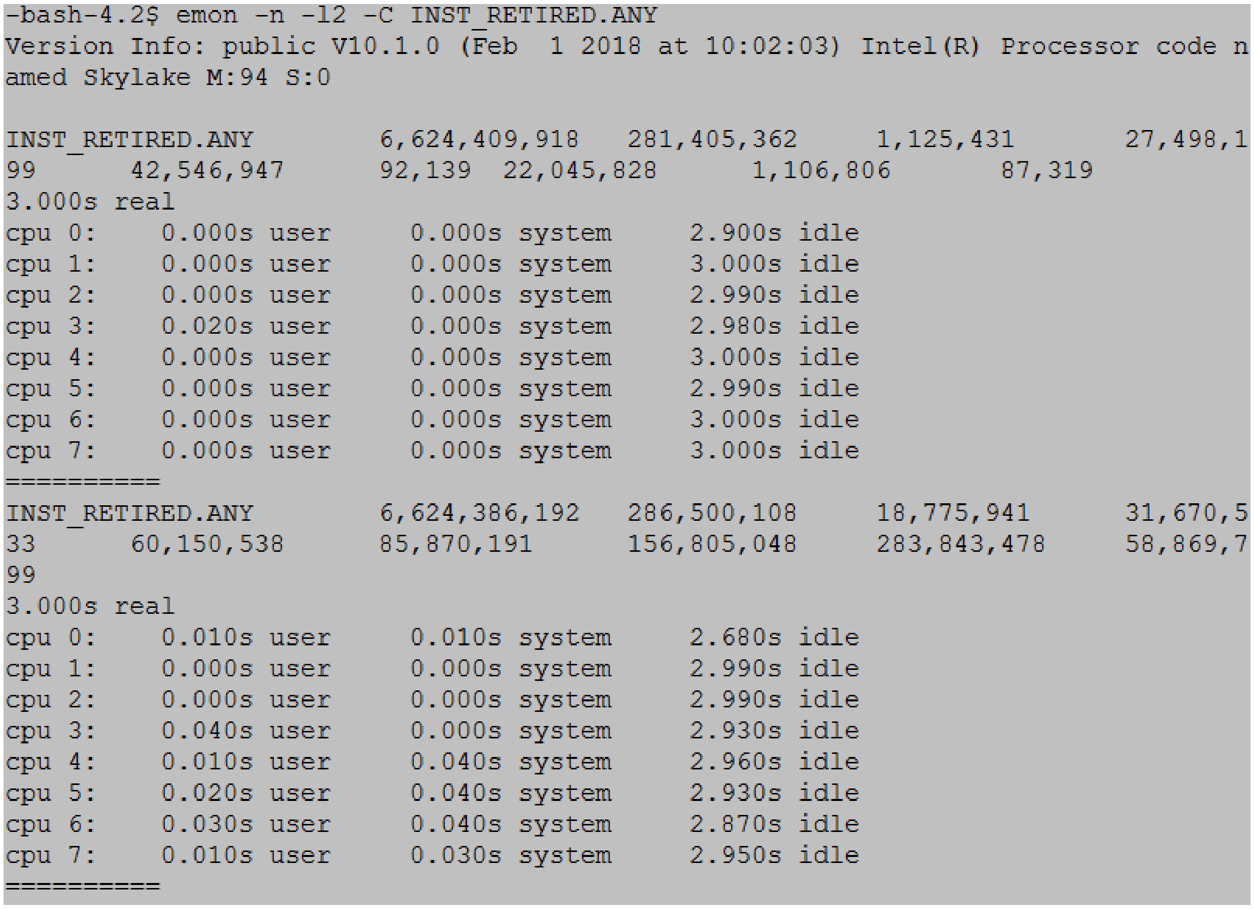
```
-bash-4.2$ emon -c -l2 -C INST_RETIRED.ANY
Version Info: public V10.1.0 (Feb  1 2018 at 10:02:03) Intel(R) Processor code n
amed Skylake M:94 S:0

02/09/2018 09:54:05.252
INST_RETIRED.ANY        6,624,416,046   302,861,568     19,593,336      401,2604
2,700,189       502,934 36,546  2,907,147       119,721
==========
02/09/2018 09:54:08.252
INST_RETIRED.ANY        6,624,200,432   291,984,537     15,850,510      14,758,5
73      50,013,329      41,477  112,279 1,227,291       2,888
==========
6.000s real
```

-d

Results are printed in formatted decimal. Formatted decimal includes comma separators. Formatted decimal is the default.

-n

Print wall clock, user, and system time for each time interval. It is only available in the command-line output.



```
-bash-4.2$ emon -n -l2 -C INST_RETIRED.ANY
Version Info: public V10.1.0 (Feb  1 2018 at 10:02:03) Intel(R) Processor code n
amed Skylake M:94 S:0

INST_RETIRED.ANY        6,624,409,918   281,405,362     1,125,431       27,498,1
99      42,546,947      92,139  22,045,828      1,106,806       87,319
3.000s real
cpu 0:    0.000s user     0.000s system     2.900s idle
cpu 1:    0.000s user     0.000s system     3.000s idle
cpu 2:    0.000s user     0.000s system     2.990s idle
cpu 3:    0.020s user     0.000s system     2.980s idle
cpu 4:    0.000s user     0.000s system     3.000s idle
cpu 5:    0.000s user     0.000s system     2.990s idle
cpu 6:    0.000s user     0.000s system     3.000s idle
cpu 7:    0.000s user     0.000s system     3.000s idle
==========
INST_RETIRED.ANY        6,624,386,192   286,500,108     18,775,941      31,670,5
33      60,150,538      85,870,191      156,805,048     283,843,478     58,869,7
99
3.000s real
cpu 0:    0.010s user     0.010s system     2.680s idle
cpu 1:    0.000s user     0.000s system     2.990s idle
cpu 2:    0.000s user     0.000s system     2.990s idle
cpu 3:    0.040s user     0.000s system     2.930s idle
cpu 4:    0.010s user     0.040s system     2.960s idle
cpu 5:    0.020s user     0.040s system     2.930s idle
cpu 6:    0.030s user     0.040s system     2.870s idle
cpu 7:    0.010s user     0.030s system     2.950s idle
==========
```

-u

Results are printed in unformatted decimal. Unformatted decimal does not include comma separators.

-x

Results are printed in hex with a leading '0x'.

Collection on Hybrid Platforms

Introduction to Hybrid Platforms

The Intel client architectures starting from 12th gen are based on a hybrid model with Performance Core (P-Core) and Efficiency Core (E-Core). Depending on the application, hybrid CPU architectures can distribute core usage more efficiently than non-hybrid architectures. P-Cores are designed to handle complex workloads while E-Cores are better suited for multi-threaded throughput and power-limited scenarios. At higher power envelopes, P-Cores can provide better performance than E-Cores. At lower power envelopes, E-Cores are more desirable. Each core type has different specifications and system configurations.

For these reasons, the P-Cores are preferred for

- Priority tasks
- Limited threading applications

while E-Cores are better suited for:

- Power-limited scenarios
- Background applications that can meet their QOS (Quality of Service) requirements on that performance

Supported Core Types

To collect samples using EMON on hybrid platforms, you must first identify the core types that are supported on your system. To do this, run:

```
emon -pmu-types
```

For example, this output indicates that two core types supported by EMON tool on the system: `p-core` and e-core

```
$ emon -pmu-types
PMU Types supported on this platform:
p-core
e-core
cbo
ncu
imc
power
```

Available Core Types

Once you have identified the core types supported by your system, find out the core types that are available. Run:

```
emon -pmu-types available
```

In this example, there are two core types available on the system: `p-core` and e-core

```
$ emon -pmu-types available
PMU Types available on this machine:
p-core
e-core
cbo
ncu
imc
power
```

Note that a core type supported by your system will not display in this output unless it is actually available on your system.

Core Type Specifications on Hybrid Platforms

Each core type has different specifications (such as cache, number of PerfMon counters etc) and system configurations on hybrid platforms.

To see the core type specification, run:

```
emon -v
```

This command displays the following types of information about supported core types:

Number of Processors per Core Type

```
$ emon -v
......
total_number_of_processors ...... 22
number_of_online_processors ...... 22
number_of_processors (P-core) ...... 12
number_of_online_processors (P-core) ...... 12
number_of_processors (E-core) ...... 10
number_of_online_processors (E-core) ...... 10
......
```

Cache Info per Core Type

```
$ emon -v
......
Cache Info (P-core):
L1 Data Cache ............ 48KB, 12-way, 64-byte line size
              2 HW threads share this cache, No SW Init Required
L1 Code Cache ............ 64KB, 16-way, 64-byte line size
              2 HW threads share this cache, No SW Init Required
L2 Unified Cache .......... 2MB, 16-way, 64-byte lin size
              8 HW threads share this cache, No SW Init Required
64-byte Prefetching

Cache Info (E-core):
L1 Data Cache ............ 32KB, 8-way, 64-byte line size
              No SW Init Required
L1 Code Cache ............ 64KB, 8-way, 64-byte line size
              No SW Init Required
L2 Unified Cache .......... 2MB, 16-way, 64-byte line size
              8 HW threads share this cache, No SW Init Required
64-byte Prefetching
......
```

Specs and Configurations per Core Type

```
$ emon -v
......
Processor Features (P-core):
number_of_selectors ...... 8
number_of_var_counters ...... 8
number_of_fixed_ctrs ..... 4
Fixed Counter Events:
counter 0 ................ INST_RETIRED.ANY
counter 1 ................ CPU_CLK_UNHALTED.THREAD
counter 2 ................ CPU_CLK_UNHALTED.REF_TSC
counter 3 ................ TOPDOWN.SLOTS
number of devices ............ 1
number_of_events ............. 595
  (Thermal Throttling) (Enabled)
  (Hyper-Threading) (Enabled)
  (DCU IP Prefetching) (Enabled)
  (DCU Streamer Prefetching) (Enabled)
  (MLC AMP Prefetching) (Enabled)
  (MLC Spatial Prefetching) (Enabled)
  (MLC Streamer Prefetching) (Enabled)
  (Cores Per Package:  6)
  (Threads Per Package: 12)
  (Threads Per COre:   2)

Processor Features (E-core):
number_of_selectors ...... 8
number_of_var_counters ...... 8
number_of_fixed_ctrs ..... 3
Fixed Counter Events:
counter 0 ................ INST_RETIRED.ANY
counter 1 ................ CPU_CLK_UNHALTED.CORE
counter 2 ................ CPU_CLK_UNHALTED.REF_TSC
number of devices ............ 1
number_of_events ............. 422
  (Thermal Throttling) (Enabled)
  (DCU IP Prefetching) (Enabled)
  (DCU Streamer Prefetching) (Enabled)
  (DCU Next Page Prefetching) (Enabled)
  (MLC Streamer Prefetching) (Enabled)
  (Cores Per Package:  5)
  (Threads Per Package: 10)
  (Threads Per Core:   1)
......
```

The information is displayed per core type with the each PMU name, such as `P-core` and E-core

Mapping Core Type to Processors

EMON collects samples for perfmon events only from applicable core types. To understand the collection result, you must first understand the core type to which each processor is mapped.

```
$ emon -v
.......
OS Processor <-> Physical/Logical Mapping
-----------------------------------------
OS Processor      Phys.Package    Core    Logical Processor    Core Type    Module
      0             0           0             0                 P-core       2
      1             0           0             1                 P-core       2
      2             0           0             0                 P-core       3
      3             0           0             1                 P-core       3
      4             0           0             0                 P-core       4
      5             0           0             1                 P-core       4
      6             0           0             0                 P-core       5
      7             0           0             1                 P-core       5
      8             0           0             0                 P-core       6
      9             0           0             1                 P-core       6
      10            0           0             0                 P-core       7
      11            0           0             1                 P-core       7
      12            0           0             0                 E-core       0
      13            0           1             0                 E-core       0
      14            0           2             0                 E-core       0
      15            0           3             0                 E-core       0
      16            0           0             0                 E-core       1
      17            0           1             0                 E-core       1
      18            0           2             0                 E-core       1
      19            0           3             0                 E-core       1
      20            0           0             0                 E-core       8
      21            0           1             0                 E-core       8
.......
```

The output indicates that processors 0-11 are `P-core` and processors 12-21 are E-core.

When the counts are generated from the collection, the counts from CPU 0-11 are for P-core and the counts from CPU 12-21 are for `E-core`.

The `emon -v` command also provides Module ID mapping to the processor if the module exists on the system.

Event Specifications

Each core type has a different perfmon event list. These events can be defined as common events or core-type specific events depends on the number of core types that have these events.

Common Events

There are events supported in multiple core types. Those events are considered as common events and are collected on all applicable processors.

To check the list of supported events per core type for all core types, run this command:

```
emon -1 [pmu name]
```

For example,

```
$ emon -1 p-core
INST_RETIRED.ANY
INST_RETIRED.PREC_DIST
BR_INST_RETIRED.ALL_BRANCHES
LONGEST_LAT_CACHE.MISS
UOPS_EXECUTED.STALLS
FRONTEND_RETIRED.L2_MISS
.......
```

```
$ emon -1 e-core
INST_RETIRED.ANY
MACHINE_CLEARS.PAGE_FAULT
MEM_UOPS_RETIRED.LOAD_LATENCY_GT_4
BR_INST_RETIRED.ALL_BRANCHES
LONGEST_LAT_CACHE.MISS
ICACHE.MISSES
ICACHE.ACCESSES
.......
```

Events that are found in events lists for both core types are **common events** such as `INST_RETIRED.ANY`,`BR_INST_RETIRED.ALL_BRANCHES`, or `LONGEST_LAT_CATCHE.MISS`.

Core-Type specific events

If the events are applicable only to certain core types, those events are considered as core-type specific events and are collected only on applicable core type processors.

To check the supported event list per core type for all core types, run this command:

```
emon -1 [pmu name]
```

For example,

```
$ emon -1 p-core
INST_RETIRED.ANY
INST_RETIRED.PREC_DIST
BR_INST_RETIRED.ALL_BRANCHES
LONGEST_LAT_CACHE.MISS
UOPS_EXECUTED.STALLS
FRONTEND_RETIRED.L2_MISS
.......
```

```
$ emon -1 e-core
INST_RETIRED.ANY
MACHINE_CLEARS.PAGE_FAULT
MEM_UOPS_RETIRED.LOAD_LATENCY_GT_4
BR_INST_RETIRED.ALL_BRANCHES
LONGEST_LAT_CACHE.MISS
ICACHE.MISSES
ICACHE.ACCESSES
.......
```

Those events which are found only in the event list for a single core type are treated as core-type specific events.

For example, the following events are exclusively `p-core` events:

- INST_RETIRED.PREC_DIST

- UOPS_EXECUTED.STALLS
- FRONTEND_RETIRED.L2_MISS

These events will be collected on `p-core` processors only.

The following events are `e-core` events:

- MACHINE_CLEARS_PAGE_FAULT
- MEM_UOPS_RETIRED.LOAD_LATENCY_GT_4
- ICACHE.MISSES
- ICACHE.ACCESSES

These events are collected on `e-core` processors only.

Event Collection

This section describes how you collect common and core-type events.

Collect Common Events

To specify common events from the event list and collect these events using EMON, run:

```
$ emon -C <common events>

for example>
$ emon -C INST_RETIRED.ANY,LONGEST_LAT_CACHE.MISS
```

Check if the events are collected from both core type processors:

```
$ emon -C INST_RETIRED.ANY,LONGEST_LAT_CACHE.MISS
Version Info: public V11.45  (Feb  8 2024 at 21:39:04) Intel(R) microarchitecture code named
Alderlake-S M:151 S:0

INST_RETIRED.ANY        4,838,886,402   1,106,189       373,020 1,340,873       563,566
1,415,054       9,200   16,497,235      1,035,595       1,026,578       149,867 137,397,821
9,205   1,749,284  24,744   723,919 29,848  1,669,973       288,156 9,207   34,626  9,206
9,206   90,802  27,316
LONGEST_LAT_CACHE.MISS  4,838,886,402   79,617  60,924  206,556 67,284  167,139 89      106,859
364,550 117,364 16,929  329,693 97      205,587 758     53,580  2,150   105,044 11,554  25
1,999   14175       2,999   839
==========
```

Because those events are common events across both `p-core` and `e-core`, counts are collected and displayed for all processors.

Core-Type Specific events Collection

- P-core events collection

To specify `p-core`-specific events from the event list and collect these events using EMON, run:

```
$ emon -C <p-core events>

for example>
$ emon -C INST_RETIRED.PREC_DIST,UOPS_EXECUTED.STALLS
```

Check if the events are collected only from `p-core` processors:

```
$ emon -C INST_RETIRED.PREC_DIST,UOPS_EXECUTED.STALLS
Version Info: public V11.45  (Feb  8 2024 at 21:39:04) Intel(R) microarchitecture code named
Alderlake-S M:151 S:0

INST_RETIRED.PREC_DIST  4,839,017,160   748,030 311,927 661,798 199,709 1,203,121       191,867
835,574 1,866,415       16,401,613      51,965  147,576,996     31,921  1,871,506       9,206
792,389 71,142      N/A     N/A     N/A     N/A     N/A     N/A     N/A     N/A
```

```
UOPS_EXECUTED.STALLS    4,839,017,160   3,044,491       1,144,642       3,081,917       842,849
5,117,158       871,912 5,757,953       29,788,795      7,793,259       112,192 13,627,971
79,848  7,537,156   42,291  3,579,026       286,040 N/A     N/A     N/A     N/A     N/A
N/A     N/A     N/A
==========
```

According to the processor mapping from emon -v output, processor 0 ~ 11 are `p-core` and 12 ~ 21 are `e-core`.

Only `p-core` processors 0 ~ 11 display samples and not applicable core type for example `e-core` processors 12 ~ 21 here display N/A

- E-core events collection

To specify `e-core`-specific events from the event list and collect these events using EMON, run:

```
$ emon -C <e-core events>

for example>
$ emon -C ICACHE.MISSES,ICACHE.ACCESSES
```

Check if the events are collected only from `e-core` processors:

```
$ emon -C ICACHE.MISSES,ICACHE.ACCESSES
Version Info: public V11.45  (Feb  8 2024 at 21:39:04) Intel(R) microarchitecture code named
Alderlake-S M:151 S:0

ICACHE.MISSES   4,838,828,820   N/A     N/A     N/A     N/A     N/A     N/A     N/A     N/A
N/A     N/A     N/A     N/A     N/A     N/A     N/A     N/A     202,043 8,004   3,179   8,196
351     13,771      332     326
ICACHE.ACCESSES 4,838,828,820   N/A     N/A     N/A     N/A     N/A     N/A     N/A     N/A
N/A     N/A     N/A     N/A     N/A     N/A     N/A     N/A     929,053 37,384  15,575  39,896
3,214   57,580      2,963   3,064
==========
```

According to the processor mapping from emon -v output, processor 0 ~ 11 are `p-core` and processor 12 ~ 21 are `e-core`.

Only `e-core` processors 12 ~ 21 display samples and not applicable core type for example `p-core` processors 0 ~ 11 here display N/A.

Collect Combination of Common and Core-Type Specific Events

To collect a combination of common events, `p-core` specific events, and e-core specific events, run:

```
$ emon -C <common events, p-core events, e-core events>

for example>
$ emon -C
INST_RETIRED.ANY,LONGEST_LAT_CACHE.MISS,INST_RETIRED.PREC_DIST,UOPS_EXECUTED.STALLS,ICACHE.MISSES
,ICACHE.ACCESSES
```

Check if all events are collected from appropriate processors like below:

```
$ emon -C
INST_RETIRED.ANY,LONGEST_LAT_CACHE.MISS,INST_RETIRED.PREC_DIST,UOPS_EXECUTED.STALLS,ICACHE.MISSES
,ICACHE.ACCESSES
Version Info: public V11.45  (Feb  8 2024 at 21:39:04) Intel(R) microarchitecture code named
Alderlake-S M:151 S:0

INST_RETIRED.ANY        4,838,712,706   1,396,384       9,206   412,857 9,206   858,794 9,214
20,170,205      918,899 1,532,879       9,207   666,507 98,113  345,820 9,207   113,560,559
60,761  2,144,426   9,206   108,957 87,373  28,499  88,063  9,207   9,206
```

```
LONGEST_LAT_CACHE.MISS  4,838,712,706   91,119  89      30,682  81      49,546  79      112,015
129,590 82,357  85      33,849  8,597   18,496  72      187,482 1,360   50,633  93      3,082
2,422   6432,125    16      29
ICACHE.MISSES   4,838,712,706   N/A     N/A     N/A     N/A     N/A     N/A     N/A     N/A
N/A     N/A     N/A     N/A     N/A     N/A     N/A     N/A     241,977 349     8,589   9,859
3,642   12,740      331     318
ICACHE.ACCESSES 4,838,712,706   N/A     N/A     N/A     N/A     N/A     N/A     N/A     N/A
N/A     N/A     N/A     N/A     N/A     N/A     N/A     N/A     1,232,926       3,168   53,812
50,364  16,850      52,998  3,072   2,904
INST_RETIRED.PREC_DIST  4,839,153,650   1,431,545       26,891  262,443 17,290  1,057,738
17,359  18,585,280      993,051 1,645,934       14,867  1,015,421       287,674 424,396 37,194
133,273,06989,148   N/A     N/A     N/A     N/A     N/A     N/A     N/A     N/A
UOPS_EXECUTED.STALLS    4,839,153,650   5,839,973       98,049  1,073,296       54,561
4,539,116       45,642  12,137,803      26,499,792      6,501,309       68,149  3,489,674
204,787 1,601,970  127,225  8,337,256       280,055 N/A     N/A     N/A     N/A     N/A
N/A     N/A     N/A
==========
```

Common events display samples to all relevent core type processors, and core type specific events display samples only on applicable core type processors and display N/A on not-applicable core type processors.

For example, LONGEST_LAT_CACHE.MISS is common event which exists in both `p-core` and `e-core`, therefore, the samples are displayed on all processors.

And INST_RETIRED.PREC_DIST is `p-core` specific event, `p-core` processors are 0 ~ 11, the samples are displayed only on `p-core` processors 0 ~ 11, and N/A was displayed on `e-core` processors 12 ~ 21 which are not-applicable processors to this event.

ICACHE.MISSES is `e-core` event, therefore the samples are displayed only on `e-core` processors 12 ~ 21 while N/A was displayed on `p-core` processors 0 ~ 11.

Collect Combination of Common and Core-Type Specific Events in Spreadsheet Topology Format

The output for same set of events collection in Spreadsheet Topology format can be collected using "-V" option

```
$ emon -C
INST_RETIRED.ANY,LONGEST_LAT_CACHE.MISS,INST_RETIRED.PREC_DIST,UOPS_EXECUTED.STALLS,ICACHE.MISSES
,ICACHE.ACCESSES -V
# SYSTEM INFORMATION FOLLOWS
# emon db : meteorlake
# num_packages : 1
# num_modules_per_package : 9
# num_cores_per_package : 16
# num_logic_processor_per_core : 2
# device p-core : num_events 4, num_unit 1
# device e-core : num_events 4, num_unit 1
# tsc_freq : 2188.80 MHz
# ufs_freq : N/A MHz
# END OF SYSTEM INFORMATION
# GROUPING INFORMATION FOLLOWS
# group 0 :
INST_RETIRED.ANY,LONGEST_LAT_CACHE.MISS,INST_RETIRED.PREC_DIST,UOPS_EXECUTED.STALLS,ICACHE.MISSES
,ICACHE.ACCESSES
# END OF GROUPING INFORMATIONS
# START OF COLLECTION
timestamp;;package0;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;
;;p-core;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;e-
core;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
epoch;timestamp;CPU0;;;;CPU1;;;;CPU2;;;;CPU3;;;;CPU4;;;;CPU5;;;;CPU6;;;;CPU7;;;;CPU8;;;;CPU9;;;;C
PU10;;;;CPU11;;;;CPU12;;;;CPU13;;;;CPU14;;;;CPU15;;;;CPU16;;;;CPU17;;;;CPU18;;;;CPU19;;;;CPU20;;;
;CPU21;;;;
;;INST_RETIRED.ANY;LONGEST_LAT_CACHE.MISS;INST_RETIRED.PREC_DIST;UOPS_EXECUTED.STALLS;INST_RETIRE
D.ANY;LONGEST_LAT_CACHE.MISS;INST_RETIRED.PREC_DIST;UOPS_EXECUTED.STALLS;INST_RETIRED.ANY;LONGEST
_LAT_CACHE.MISS;INST_RETIRED.PREC_DIST;UOPS_EXECUTED.STALLS;INST_RETIRED.ANY;LONGEST_LAT_CACHE.MI
SS;INST_RETIRED.PREC_DIST;UOPS_EXECUTED.STALLS;INST_RETIRED.ANY;LONGEST_LAT_CACHE.MISS;INST_RETIR
ED.PREC_DIST;UOPS_EXECUTED.STALLS;INST_RETIRED.ANY;LONGEST_LAT_CACHE.MISS;INST_RETIRED.PREC_DIST;
UOPS_EXECUTED.STALLS;INST_RETIRED.ANY;LONGEST_LAT_CACHE.MISS;INST_RETIRED.PREC_DIST;UOPS_EXECUTED
.STALLS;INST_RETIRED.ANY;LONGEST_LAT_CACHE.MISS;INST_RETIRED.PREC_DIST;UOPS_EXECUTED.STALLS;INST_
RETIRED.ANY;LONGEST_LAT_CACHE.MISS;INST_RETIRED.PREC_DIST;UOPS_EXECUTED.STALLS;INST_RETIRED.ANY;L
ONGEST_LAT_CACHE.MISS;INST_RETIRED.PREC_DIST;UOPS_EXECUTED.STALLS;INST_RETIRED.ANY;LONGEST_LAT_CA
CHE.MISS;INST_RETIRED.PREC_DIST;UOPS_EXECUTED.STALLS;INST_RETIRED.ANY;LONGEST_LAT_CACHE.MISS;INST
_RETIRED.PREC_DIST;UOPS_EXECUTED.STALLS;INST_RETIRED.ANY;LONGEST_LAT_CACHE.MISS;ICACHE.MISSES;ICA
CHE.ACCESSES;INST_RETIRED.ANY;LONGEST_LAT_CACHE.MISS;ICACHE.MISSES;ICACHE.ACCESSES;INST_RETIRED.A
NY;LONGEST_LAT_CACHE.MISS;ICACHE.MISSES;ICACHE.ACCESSES;INST_RETIRED.ANY;LONGEST_LAT_CACHE.MISS;I
CACHE.MISSES;ICACHE.ACCESSES;INST_RETIRED.ANY;LONGEST_LAT_CACHE.MISS;ICACHE.MISSES;ICACHE.ACCESSE
S;INST_RETIRED.ANY;LONGEST_LAT_CACHE.MISS;ICACHE.MISSES;ICACHE.ACCESSES;INST_RETIRED.ANY;LONGEST_
LAT_CACHE.MISS;ICACHE.MISSES;ICACHE.ACCESSES;INST_RETIRED.ANY;LONGEST_LAT_CACHE.MISS;ICACHE.MISSE
S;ICACHE.ACCESSES;INST_RETIRED.ANY;LONGEST_LAT_CACHE.MISS;ICACHE.MISSES;ICACHE.ACCESSES;INST_RETI
RED.ANY;LONGEST_LAT_CACHE.MISS;ICACHE.MISSES;ICACHE.ACCESSES;
1684966389500;6569377555;8979741;470077;109978866;2694901;6414492;56266;33811194;2694901;483284;1
08215;22047810;69963;15393;543;254430;91892;147192;36376;6143424;0;15391;439;292578;0;49059;7496;
1436868;12617;19924;608;315744;12617;32739;1046;660948;10492;37611;6632;1224936;10492;167034;6715
;4024518;26116;170488;39450;5492808;26116;15612;755;6569;29837;1059438;400228;1346644;2821052;323
31;6413;29063;72590;15288;624;6081;27984;15287;880;6620;28762;155176;16128;106452;319045;2625821;
27885;108750;1092889;15319;959;8323;29185;25243;2963;10368;44538;93983;51623;158700;282613;
```

Spreadsheet Topology format additionally provides system information as well as grouping information.

The header indicates the order of sample data in the output.

Resource Director Technology (RDT) Collection

Introduction

Intel® Resource Director Technology (Intel® RDT) is a set of monitoring capabilities that you can use to measure shared resource metrics such as L3 cache occupancy in each logical processor.

The Resource Monitoring ID (RMID) is used to monitor the shared resources. The RMID provides a layer of abstraction between the software thread and logical processors. Each software thread is assigned to a unique RMID. The RMID can be assigned to a single logical processor or multiple logical processors (through IA32_PQR_ASSOC_MSR) for monitoring.

Operating Technologies

The operations of Intel® RDT are governed by two technologies:

Two feature within the monitoring feature set provided are like below:

- **Cache Monitoring Technology (CMT)**: This allows an operating system, hypervisor, or similar system management agent to determine the usage of cache by applications running on the platform. The associated event in EMON is `UNC_CMT_L3_CACHE_OCCUPANCY`.
- **Memory Bandwidth Monitoring (MBM)**: This is used to monitor the bandwidth from one level of the cache hierarchy to the next. The associated event in EMON is `UNC_MBM_TOTAL_EXTERNAL_BW`, `UNC_MBM_LOCAL_EXTERNAL_BW`.

You can find more information about these technologies in Chapter 17.16 of the Intel® Software Developer Manual.

Additionally, EMON provides the RMID association and RDT allocation through these events:

- `UNC_RDT_PQR_ASSOC` - bit 0:9 represents RMID and bit 32:63 represents CLOS
- `UNC_CAT_L2_MASK` - represents L2 cache allocation capacity associated with the COS on each logical processors.
- `UNC_CAT_L3_MASK` - represents L3 cache allocation capacity associated with the COS on each logical processors.

For more information, see these chapters in the Intel® Software Developer Manual:

- Monitoring Resource (RMID) Association - Chapter 17.16.6
- Cache Allocation Technology Architecture - Chapter 17.17.1

RDT Support Information

To see support information for Intel RDT on your system, run:

```
emon -v
```

For example,

```
$ emon -v
.......
RDT HW Support:
    L3 Cache Occupancy          : Yes
    Total Memory Bandwidth      : Yes
    Local Memory Bandwidth      : Yes
    L3 Cache Allocation         : Yes
    L2 Cache Allocation         : No
    Highest Available RMID      : 175
    Sample Multiplier           : 90112
    Number of MBA CLOS          : 15
```

```
......
```

Supported RDT Events

SEP determines the support for each RDT event. To see a list of these events, run:

```
emon -1 rdt
```

For example,

```
$ emon -1 rdt
UNC_CMT_L3_CACHE_OCCUPANCY
UNC_MBM_TOTAL_EXTERNAL_BW
UNC_MBM_LOCAL_EXTERNAL_BW
UNC_RDT_PQR_ASSOC
UNC_CAT_L2_MASK
UNC_CAT_L3_MASK
```

Collect RDT Events

To collect RDT events, run:

```
emon -C <RDT Event List>
```

For example,

```
$ emon -C <RDT Event List>

$ emon -C
UNC_CMT_L3_CACHE_OCCUPANCY,UNC_MBM_TOTAL_EXTERNAL_BW,UNC_MBM_LOCAL_EXTERNAL_BW,UNC_RDT_PQR_ASSOC,
UNC_CAT_L2_MASK,UNC_CAT_L3_MASK
```

RDT Standalone Mode

To profile cache usage by hardware core, include the `-rdt-auto-rmid` option. The EMON tool assigns the core ID for each core as the RMID.

```
$ emon -C <RDT Event List> -rdt-auto-rmid
```

Logging Options

--dump-driver-log [file_name]

Dump the contents of the sampling driver's internal log to the given file in binary format. Default file name is `driver_log.dump` if none specified.

```
emon --dump-driver-log
```

--decode-driver-log [input_file]

Decode the log buffer dump to text format. Default file to decode would be `driver_log.dump` if none is specified.

```
emon --decode-driver-log
```

--extract-driver-log <input core dump> [output file]

Identifies and extracts the most recent instance of the driver log from the specified uncompressed core dump into the output file. Default output file is `driver_log.dump` if none specified.

```
emon --extract-driver-log ./core.dump
```

Other Options

-experimental

Experimental events are those events that have not been validated in hardware. When used with `emon -1`, all available experimental events are displayed along with regular events. To list experimental along with regular events, use the following command:

```
emon -1 -experimental
```

To run collection on experimental events, use:

```
emon -C "<EVENT1,EVENT2>" -experimental
```

--per-cpu-tsc

Display timestamp counter value on each core.

```
$ emon --per-cpu-tsc -C CPU_CLK_UNHALTED.REF_TSC
Version Info: public V11.45  (Feb  8 2024 at 21:39:04) Intel(R) microarchitecture code named
Coffeelake M:158 S:10

TSC_VALUE       9,577,754,242   9,577,754,242   9,577,755,072   9,577,754,264   9,577,530,528
9,577,754,407   9,577,755,030   9,577,754,224   9,577,755,179   9,577,754,288   9,577,557,202
9,577,754,341       9,577,754,961
CPU_CLK_UNHALTED.REF_TSC        9,577,754,242   9,601,004       3,586,611       6,761,321
4,558,575       20,537,062      12,168,170      10,449,677      14,594,489      37,408,112
8,231,636  17,283,882       41,970,943
==========
3.000s real
```

--per-cpu-absolute-tsc

This option prints absolute timestamp value on each core. This feature helps to correlate with other types of data collected on the system.

```
$ emon --per-cpu-absolute-tsc -C CPU_CLK_UNHALTED.REF_TSC
Version Info: public V11.45  (Feb  8 2024 at 21:39:04) Intel(R) microarchitecture code named
Coffeelake M:158 S:10

TSC_ABSOLUTE_VALUE      5,436,596,628,262,328   5,436,596,628,262,328   5,436,596,628,262,384
5,436,596,628,262,304   5,436,596,628,061,615   5,436,596,628,262,328   5,436,596,628,263,003
5,436,596,628,262,360       5,436,596,628,262,429   5,436,596,628,262,283
5,436,596,628,036,828   5,436,596,628,262,376   5,436,596,628,262,900
CPU_CLK_UNHALTED.REF_TSC        5,436,596,628,262,328   18,724,538      46,608,520
148,557,143     49,034,174      14,900,522      8,415,309       24,188,311      5,936,854
3,759,511       443,525,607 1,960,952       25,086,460
==========
3.000s real
cpu 0:    0.000s user     0.000s system     3.000s idle
cpu 1:    0.000s user     0.000s system     2.990s idle
cpu 2:    0.030s user     0.010s system     2.960s idle
cpu 3:    0.000s user     0.000s system     2.990s idle
```

-verbose

Display EMON output in verbose mode.

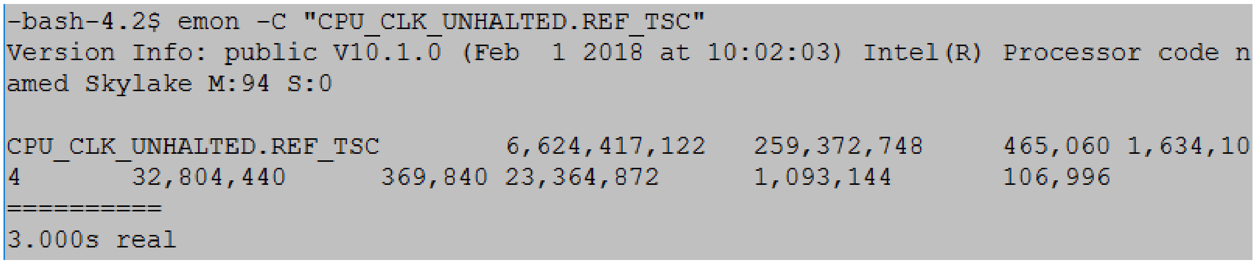
Examples

3

This chapter describes the most common EMON use cases.

Basic

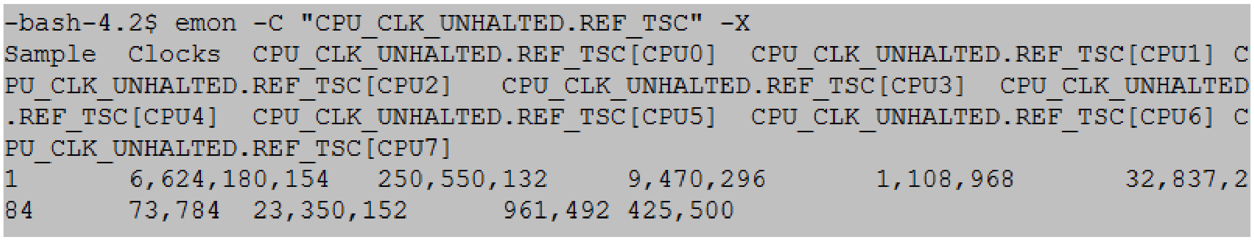
This is the most basic EMON command to run a collection.



```
-bash-4.2$ emon -C "CPU_CLK_UNHALTED.REF_TSC"
Version Info: public V10.1.0 (Feb  1 2018 at 10:02:03) Intel(R) Processor code n
amed Skylake M:94 S:0

CPU_CLK_UNHALTED.REF_TSC        6,624,417,122   259,372,748     465,060 1,634,10
4       32,804,440      369,840 23,364,872      1,093,144       106,996
==========
3.000s real
```

If not otherwise specified, EMON will monitor once for an interval of 3 s. To change either the interval length or the number of intervals (or loops), use the `-t` or `-l` options, respectively.

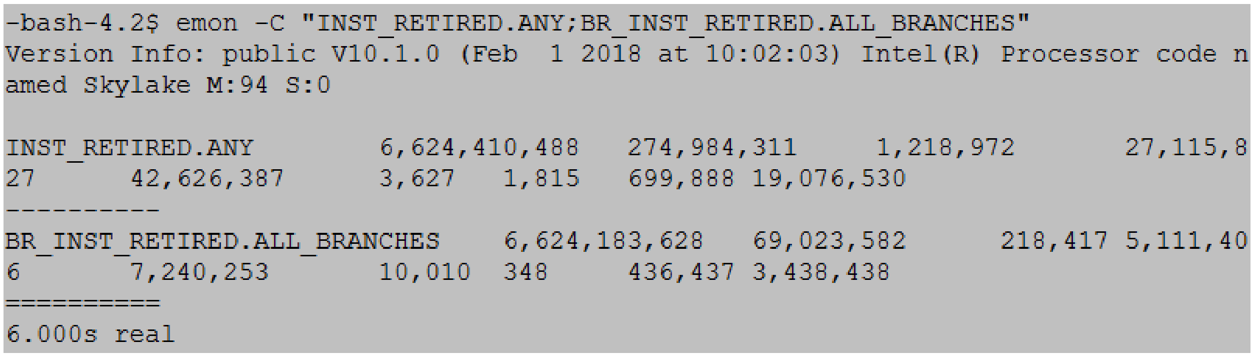
The basic command creates the data output in quiet mode, which means a minimal amount of output. To print out the headers for importing into a spreadsheet, specify the spreadsheet mode with the `-X` flag.



```
-bash-4.2$ emon -C "CPU_CLK_UNHALTED.REF_TSC" -X
Sample  Clocks  CPU_CLK_UNHALTED.REF_TSC[CPU0]  CPU_CLK_UNHALTED.REF_TSC[CPU1] C
PU_CLK_UNHALTED.REF_TSC[CPU2]   CPU_CLK_UNHALTED.REF_TSC[CPU3]  CPU_CLK_UNHALTED
.REF_TSC[CPU4]  CPU_CLK_UNHALTED.REF_TSC[CPU5]  CPU_CLK_UNHALTED.REF_TSC[CPU6] C
PU_CLK_UNHALTED.REF_TSC[CPU7]
1       6,624,180,154   250,550,132     9,470,296       1,108,968       32,837,2
84      73,784  23,350,152      961,492 425,500
```

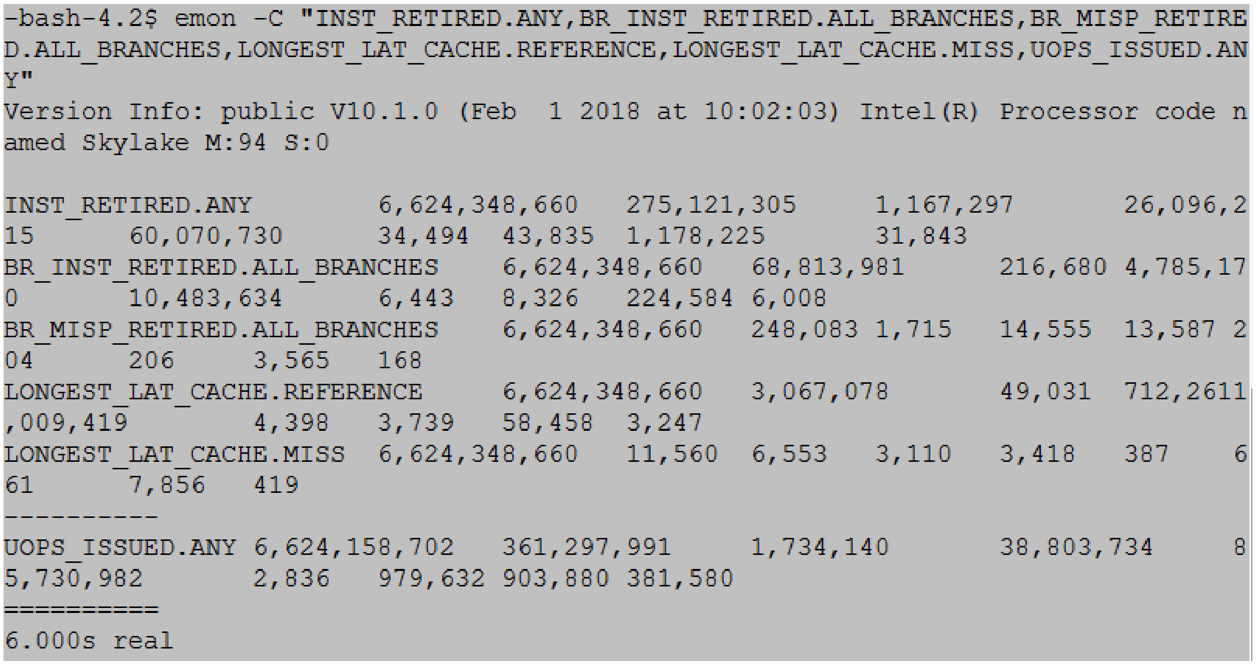
Multi-group Core Events

Events can be broken in to multiple groups forcibly through command line or automatically scheduled in to multiple groups by the tool due to hardware counter restrictions. EMON command launches multiple groups forcibly as shown below (note the semicolon `(;)` instead of comma `(,))`:



```
-bash-4.2$ emon -C "INST_RETIRED.ANY;BR_INST_RETIRED.ALL_BRANCHES"
Version Info: public V10.1.0 (Feb  1 2018 at 10:02:03) Intel(R) Processor code n
amed Skylake M:94 S:0

INST_RETIRED.ANY        6,624,410,488   274,984,311     1,218,972       27,115,8
27      42,626,387      3,627   1,815   699,888 19,076,530
----------
BR_INST_RETIRED.ALL_BRANCHES    6,624,183,628   69,023,582      218,417 5,111,40
6       7,240,253       10,010  348     436,437 3,438,438
==========
6.000s real
```

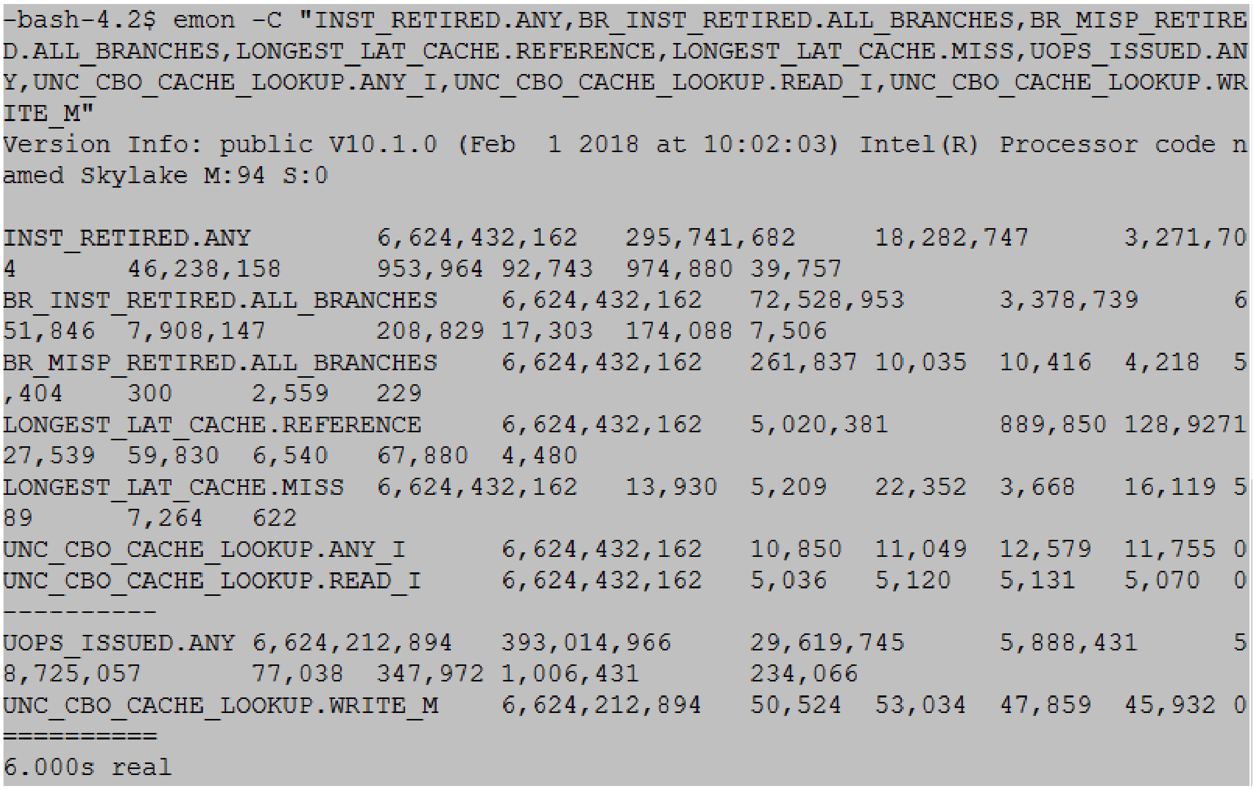
Assuming a CPU core has four general purpose (GP) counters, the tool can program only four GP events in a single iteration of event collection. The remaining events will be moved into new groups. EMON performs multiple runs for each group. In the following example, the GP event `UOPS_ISSUED.ANY` is scheduled in a second run.



```
-bash-4.2$ emon -C "INST_RETIRED.ANY,BR_INST_RETIRED.ALL_BRANCHES,BR_MISP_RETIRE
D.ALL_BRANCHES,LONGEST_LAT_CACHE.REFERENCE,LONGEST_LAT_CACHE.MISS,UOPS_ISSUED.AN
Y"
Version Info: public V10.1.0 (Feb  1 2018 at 10:02:03) Intel(R) Processor code n
amed Skylake M:94 S:0

INST_RETIRED.ANY        6,624,348,660   275,121,305     1,167,297       26,096,2
15      60,070,730      34,494  43,835  1,178,225       31,843
BR_INST_RETIRED.ALL_BRANCHES    6,624,348,660   68,813,981      216,680 4,785,17
0       10,483,634      6,443   8,326   224,584 6,008
BR_MISP_RETIRED.ALL_BRANCHES    6,624,348,660   248,083 1,715   14,555  13,587 2
04      206     3,565   168
LONGEST_LAT_CACHE.REFERENCE     6,624,348,660   3,067,078       49,031  712,2611
,009,419        4,398   3,739   58,458  3,247
LONGEST_LAT_CACHE.MISS  6,624,348,660   11,560  6,553   3,110   3,418   387    6
61      7,856   419
----------
UOPS_ISSUED.ANY 6,624,158,702   361,297,991     1,734,140       38,803,734     8
5,730,982       2,836   979,632 903,880 381,580
==========
6.000s real
```

Multi-group Core and Uncore Events

The number of events programmed in each group for a device depends on available counters on that device. For example, group 0 could have 4 GP events on a core, 2 GP events per CBO unit, 1 GP event per PCU unit, and so on. In the following example, the first group has 4 GP events on a core and 2 GP events on CBO. The remaining core and CBO events are scheduled in the next group.



```
-bash-4.2$ emon -C "INST_RETIRED.ANY,BR_INST_RETIRED.ALL_BRANCHES,BR_MISP_RETIRE
D.ALL_BRANCHES,LONGEST_LAT_CACHE.REFERENCE,LONGEST_LAT_CACHE.MISS,UOPS_ISSUED.AN
Y,UNC_CBO_CACHE_LOOKUP.ANY_I,UNC_CBO_CACHE_LOOKUP.READ_I,UNC_CBO_CACHE_LOOKUP.WR
ITE_M"
Version Info: public V10.1.0 (Feb  1 2018 at 10:02:03) Intel(R) Processor code n
amed Skylake M:94 S:0

INST_RETIRED.ANY        6,624,432,162   295,741,682     18,282,747      3,271,70
4       46,238,158      953,964 92,743  974,880 39,757
BR_INST_RETIRED.ALL_BRANCHES    6,624,432,162   72,528,953      3,378,739      6
51,846  7,908,147       208,829 17,303  174,088 7,506
BR_MISP_RETIRED.ALL_BRANCHES    6,624,432,162   261,837 10,035  10,416  4,218  5
,404    300     2,559   229
LONGEST_LAT_CACHE.REFERENCE     6,624,432,162   5,020,381       889,850 128,9271
27,539  59,830  6,540   67,880  4,480
LONGEST_LAT_CACHE.MISS  6,624,432,162   13,930  5,209   22,352  3,668   16,119 5
89      7,264   622
UNC_CBO_CACHE_LOOKUP.ANY_I      6,624,432,162   10,850  11,049  12,579  11,755 0
UNC_CBO_CACHE_LOOKUP.READ_I     6,624,432,162   5,036   5,120   5,131   5,070  0
----------
UOPS_ISSUED.ANY 6,624,212,894   393,014,966     29,619,745      5,888,431      5
8,725,057       77,038  347,972 1,006,431       234,066
UNC_CBO_CACHE_LOOKUP.WRITE_M    6,624,212,894   50,524  53,034  47,859  45,932 0
==========
6.000s real
```

Help and Troubleshoot

4

This chapter provides helpful tips and troubleshooting guidance.

Getting Started With EMON

To get started with EMON:

1. Identify hardware events of interest using `emon -1/-?` options.

   > **NOTE** For details on event descriptions, see Intel® Software Developer’s Manual (Intel® SDM) documentation. Events mentioned in the examples in this guide may not work on all platforms since each platform has its own event lists.

2. Identify processor and memory configuration using `emon -v`.
3. Refer to the applicable sections in this document or use `emon -h` to understand the available tool options and example usages.

Discarded Events

The following situations could result in discarded events:

- An event could be discarded if it is not available on the platform. If an event is discarded due to this reason, the event will not be displayed by `emon -1`.
- An event could be discarded if the system does not come with the device types that support the event. For example, if a system does not come with FPGA units, FPGA events would be discarded.
- If it is a private event and needs special access privileges. In such a case, the event will not be displayed by `emon -1`. By using an non-disclosure agreement (NDA) release package, this problem can be resolved.

Experimental Events

Some events are available as experimental events if they are not verified in the hardware. These events are not displayed by `emon -1`. To get event list along with available experimental events use, `emon -1 -experimental` or `emon -1 -all`. To collect data on experimental events, use `emon -C -experimental`.

Deprecated Events

Certain events are marked deprecated by the tool. EMON will stop supporting deprecated events in future product releases. The tool provides replacement suggestions in place of deprecated events. To obtain a list of deprecated events, use `emon -?` and look for "deprecated" string.