Using the VTune(TM) Performance Analyzer Sampling Collector for Mobile Internet Device (MID)

User's Guide

Document Number: 318121-004

Revision: 1.4

World Wide Web: http://www.intel.com

Contents

Disclaimer and Legal Information

1 Overview

The VTune™ Performance Analyzer sampling data collector for Mobile Internet Device (MID) is a standalone command-line tool that provides the event-based sampling (EBS) functionality on a local system. The hardware-based sampling is a low-overhead, system-wide profiling that helps to identify which modules and functions are consuming the most time, giving a detailed look at the operating system and application. This tool enables you to configure the data collection, perform the system-wide profiling, and store the results in a file.

The goal of this guide is to introduce you to all the sampling collector command-line options and explain how to use them to get more useful results.

2 Sampling Collector Command-line Options

This chapter details the command-line options of the VTune analyzer sampling collector for MID.

2.1 Generic Collector Options

[-d | -duration <in seconds>]

Specify duration for the sampling collection

Use the `-d` option to specify duration for the sampling collection. Default is 20 seconds. Set duration to zero to run collection for an indefinite amount of time until it is stopped explicitly with the `sep -stop` command.

[-nb | -non-blocking]

Switch to non-blocking mode

Use the `-nb` option to switch the sampling command-line collector to non-blocking mode. The collector starts in the background. You regain control after data collection starts. When the sampling collector is in background the default behavior is “blocking”.

[-c | -count]

Count selected events

Use the `-c` option to count the selected events. The event counting results are available in the output file with a .txt extension. This is the format of the `XXX.txt` file:

```
event, cpu #, count, duration
```

Where each event or processor has a separate line.

[-ce | -count-emon]

Count selected events in EMON style

```
[-ce | -count-emon]
```

Use the –ce option to count the selected events and output in EMON style, using the equivalent of the `-q` and `-u` switches in EMON (terse mode). Use the `-out` option to specify the output file. The format of the output is:

```
Event    Timestamp        Count (cpu0)                    Count(cpuN)
```

Where each event has its own line and the count for each processor is printed after the current timestamp. This is followed by a line with the total/user/system/idle times taken by each run. All fields are tab separated. Events of different runs are be separated by a row of dashes (`-`).

You can also use this option with the `-si` option to perform interval counting. The value specified after the `-si` option is the length of the interval (in milliseconds). When interval counting is specified, the output for each interval is in the same format as specified above, except the count value for each processor is the counts collected since the last interval (not cumulative). Each interval is separated by a row of equal signs (`=`). After the counting duration has elapsed, the cumulative count values are displayed at the end, with an additional row of dashes (`-`) separating between the last interval and the cumulative values.

[-rc | -rerun-based-on-count]

Run to calculate Sample After value

Use the `-rc` option to set the sampling collector to generate event count information and uses that information to calculate a reasonable Sample After Value (SAV) for each event.

After calculating the SAVs, the sampling collector reruns the data collection, using the calculated SAVs, and collects samples.

[-si | -sampling-interval <interval in milliseconds>]

Specify the amount of time between samples

Use the `-si` option to specify the amount of time between collected samples. In Time-based sampling (TBS), the actual time between samples is dictated by OS dependencies. In Event-based sampling (EBS) mode you can use the `-rerun-based-on-count` option to have the sampling collector determine a sample-after value, which is used to calculate average number of samples to collect per second, per event. Default is 1 ms.

This option can also be used to enable interval counting in EMON output mode. See –ce |count-emon option for more information.

NOTE: TBS is not supported on Linux*.

[-sb | -sample-buffer-size <size in kilobytes>]

Specify the buffer-size for storing sample data

Use the `-sb` option to specify size of buffer to store samples in before saving them to disk. Default is 2000KB.

[-sd | -sampling-delay <delay in seconds>]

Specify delay of data collection

Use the –sd option to specify number of seconds to delay sampling while your application is being executed. Default is 0 sec. Start delay is a separate time interval that is not a part of duration. For example, if you have an Activity with duration of 60 seconds and a start delay of 10 seconds, the sampling collector starts collecting samples after 10 seconds and runs for 60 seconds taking a total time of 70 seconds.

[-msc | -max-samples-to-collect <maximum number of samples to collect>]

Specify a total of samples to collect before stopping data collection

Use `-msc` to specify the total number of samples to collect before stopping data collection. Default is 'no maximum'.

[-sm | -sampling-method <ebs|tbs>]

Select event or time-based sampling

Use the `-sm` option to choose between time-based (TBS) and event-based sampling (EBS). Default is ebs.

NOTE: TBS is not supported on Linux*.

[-sp | -start-paused]

Start data collection in paused mode

Use the –sp option to start data collection in pause mode. To resume collection use the `-resume` option.

[-cm | -cpu-mask <"text to specify the cpu mask">]

Specify what processors to collect data from

Use the –cm option to specify a cpu-mask that defines the processors that you wish to collect data from. Enter the processor numbers or processor ranges separated by commas. For example:

```
-cm <2-5,10, 12-14>
```

In this example the only following processors are sampled: 2, 3, 4, 5, 10, 12, 13, 14

[-out | -output-file <file name>]

Specify the file name for the output file

Specify the base file name of the output file where the data is collected. The sampling collector appends an extension that is appropriate for the type of sampling/counting that is collected. For a sampling run it will append .tb5 and for a counting run it will append a .txt).

If the option is not specified, then the output file(s) name will be chosen randomly. In the case of multiple runs, in addition to the above, the output sampling file name is extended to indicate each run number.

For example, the name `foo` is saved as `foo_001.tb5`, `foo_002.tb5`.

[-of | -options-from-file <file name>]

Read the sampling collector options from a file

Use the `-of` to specify a file from which to read the sampling options. The collector reads the options from the specified file and applies them.

The options specified in the file use the command line options. The options can be specified in the same line or multiple lines.

For example, this is the content of `my_clocks.txt`:

```
-d 10
-ec CPU_CLK_UNHALTED.CORE:sa=1000000
-out clock_out
```

You can get the same results using the following two commands:

```
sep -start -of my_clocks.txt
```

Or,

```
sep -start -d 10 -ec CPU_CLK_UNHALTED.CORE:sa=1000000 -out clock_out
```

NOTE: The command-line options take precedence over the options from a file.

[-app <full-path-to-the-application>]

Specify the application to be launched for data collection

Specify the application to be launched with the sampling collector. You need to specify the full path to the application.

The application will not be launched if the sampling session is configured to have multiple runs, either through the `-rerun…|-rc` command or by having events that cannot be collected simultaneously in a single run.

NOTE: The `-d | -duration` option is not supported with this option. The sampling data collection continues indefinitely until the launched application terminates or the collector is stopped explicitly with the `sep -stop` command.

[-em | -event-multiplexing]

Enable event multiplexing

Use the `-em` option to enable event multiplexing. Event multiplexing is the ability to sample multiple groups of events within a single sampling run.

The event multiplexing works in the Timer mode. When using this mode, the sampling collector collects data at specific time intervals. You can define the time interval (`dts`) and

the list of events to be counted. If `dts` is not specified, the default is 50 milliseconds. The minimal `dts` value is 10 milliseconds.

For example:

```
sep -start -em dts=100 -ec "INST_RETIRED.ANY_P", "BR_INST_RETIRED.ANY",
"CPU_CLK_UNHALTED.CORE", "L2_LINES_IN.SELF.ANY"
```

This sampling run collects samples of the following events every 100 milliseconds: INST_RETIRED.ANY_P. BR_INST_RETIRED.ANY, CPU_CLK_UNHALTED.CORE, and L2_LINES_IN.SELF.ANY.

2.2 Event Specific Options

This section describes the options for selecting events.

[-ec | -event-config]

Configure the events that are sampled

Event configuration options begin with `–ec | -event-config` switch. Specify the event(s) to monitor and embed the event names within single quotes(‘).

The `[:modifier=val]` option enables you to specify individual event modifiers along with the respective values for a given platform.

The modifiers can be generic to an event as well as specific to a constraint (or an event qualifier). The constraint specific special modifiers appear after `[/constraint=]`. The modifier values can be in decimal or hexadecimal format. Only specific modifiers accept the value as a string.

Each event specification is delimited by a comma (,).

[-dc | -data-config]

Configure the data that is collected

A `–dc | -data-config` clause can be specified in the `–ec` switch. The `–dc` clause allows you to specify an additional list of data values to be collected with every sample. An example of such data is ‘`power-states`’. Specifying ‘`-dc power-states`’ enables the collection of power states on systems support power-states.

If the `–dc` clause is specified with an event-name in the switch, the optional data values are collected only when that particular event is sampled. But if `-dc` is specified without an

event name, it applies to all the events sampled during a collection. The current version of the VTune analyzer sampling collector supports only the latter usage mode.

The `-dc` clause does not affect data values which are collected by default during a sampling collection. For example, ip address and timestamp values are always collected at every sample.

Syntax

```
[-ec | -event-config  [-dc | -data-config <optional-data1>,<optional-data2>…
] '<event
name1>':modifier1=val:modifier2=val/constraint1={:modifier3=val:modifier4=va
l}, '<event-name2>'...]
```

Optional Data Values

`power-states`

Enables collection of power states on systems that support Enhanced Intel® Speedstep Technology.

2.2.1 Generic Event Modifiers

:sa | sample-after = <sample after value>

Sample After Value (SAV) for the event, indicates the number of events after which a sample is collected. See Calculating Sample After Value for information on how to compute SAV. Following is an explanation of how to compute SAV. The following are modifiers by processor.

The values for the following attributes can be either in hex or decimal format.

2.2.2 Event Modifiers for P6 Processor Family

The following table lists the event modifiers for the P6 processor family and provides a short description of each modifier.

Table 1: Event Modifiers for the P6 Processor Family

Modifier	Description
`Event_Select=<value>`	Selects event to be monitored
`UMASK=<value>`	Unit Mask, further qualifies the selected event.
`USR=<yes/no>`	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 OS flag.
`OS=<yes/no>`	Specifies that events are counted only when the processor is operating at privilege level 0. This flag can be used in conjunction with the USR flag.
`E=<yes/no>`	Enables edge detection of events.
`PC=<yes/no>`	Enables toggling PM*i* pins.
`INT=<yes/no>`	Enables APIC interrupt.
`EN=<yes/no>`	Enables performance monitoring counters.
`INV=<yes/no>`	Inverts the result of the counter-mask comparison when set, so that both greater than and less than comparisons can be made.
`CMASK=<yes/no>`	Enables counter mask.
`PRECISE=<yes/no>`	Enables the PEBS feature for the PEBSable event.

Table 2: Event Modifiers for the Intel(R) Atom(TM) processor

Modifier	Description
`ANYTHR=<yes/no>`	Enables counting to continue while in any processor thread on the core is in C0 power state.

3 Using the VTune Analyzer Sampling Collector for Mobile Internet Device (MID)

This chapter explains how to use the command-line sampling collector for Mobile Internet Device (MID).

3.1 Starting Data Collection

When a sampling collection is started (using `sep -start`) the default behavior of this call is blocking. This means that the control is returned back to you only after the sampling collection finishes.

3.1.1 Output

A successful EBS run gives the following message:

```
SEP is collecting samples based on the following events <event names
separated by a comma >
```

An unsuccessful run gives the following message:

```
SEP failed to start sampling collection due to one of the following
reason(s): <error message>
```

3.2 Duration Considerations

The following sections explain several considerations on the duration of a sampling activity.

3.2.1 Pause

When a sampling run is paused (using sep -pause), the duration of the run does not change. For example, if a sampling run is started for duration of 60 seconds and it is paused after approximately 20 seconds, then the sampling activity will still complete after 60 seconds, but the data is only collected during the first 20 seconds before it was paused.

3.2.2 Start Delay

Start delay is a separate time interval that is not a part of duration. For example, if you have an activity with duration of 60 seconds and a start delay of 10 seconds, then the sampling collector will start collecting samples after 10 seconds and run for 60 seconds taking a total time of 70 seconds.

3.2.3 Zero Duration and Multiple Runs

When you set zero duration, this means that the sampling collector continues to run sampling indefinitely. In this case the sampling activity should not have multiple runs.

The sampling collector needs to run the collection multiple times for the following reasons:

- The events selected cannot be grouped in a single run. It requires multiple runs to sample all the events.
- `-rc | -rerun-based-on-count` option is chosen. This at least runs the collection twice; once to get a count of the events seen during the run and the second for the actual data collection using the event counts from the first run to help generate sample after values.

Therefore, zero duration is not allowed in multiple-run collections.

3.3 Data Collection Considerations

3.3.1 Starting Applications with the Sampling Collector

The sampling collector enables launching an application using `-app` switch. The sampling collector starts the workload and collects data until the application finishes (or is forcibly terminated) or the collector is stopped explicitly using `sep -stop` command.

In general, an activity can be controlled using one of the following methods, but not both:

- Duration: the sampling collector runs for the specified duration. An asynchronous session can be started by specifying zero duration.
- Application: When launched an application with the sampling collector, the collector will run until the app is running. The data collection session will only be stopped once the app terminates or the session is explicitly stopped with `sep -stop`.

Following are some additional restrictions of this usage model:

- When the sampling collector is explicitly stopped, it will not terminate the application on your behalf. In general, the sampling collector will only launch the application but never forcibly terminate it.

- An application can only be specified if the collector is configured to have a single run. In the case of multiple runs (either due to `–rc |-re-run-based-on-count` option or selected events cannot be grouped in one run), the application launch option will not be valid.

3.3.2 Calculating Sample After Value

The VTune analyzer sampling collector initially computes a default sample after value (SAV) for the default events (CPU_CLK_UNHALTED.CORE or CPU Cycles and retired instructions) as follows:

SAV for default events = CPU frequency * sample interval in microseconds.

To calculate the sample after value for any event, do the following:

1. Calculate the targeted (or expected) number of samples:
   Targeted Number of Samples =
   (Sampling Duration / Sampling Interval) * Number of processors
2. Calculate the average number of event counts for a single processor
   Avg. number of event counts = Total event counts across all CPUs / Number of CPUs
3. Finally, compute the sample after value (SAV) as
   Sample After Value (SAV) = Average number of event counts (as in 2) / Targeted number of samples (as in 1).

The minimum value for SAV is 1. The sample after value should not be zero or a negative value.

3.3.3 Event Count and Re-run Based on Count

When the `-count` option is specified, a file with `*.txt` extension (`XXX.txt`) is created and all of the counts for that `sep -start -count` session (which may have multiple runs) are appended to the file. Even when a sampling activity generates multiple tb5 files, it will always generate only one `XXX.txt` file, per sampling session. A duration of 0 is allowed for the `-count` (without the `[-rc | -rerun-based-on-count]` option) and a single run as the `XXX.txt` output file can be written at the time of `sep -stop`.

The zero duration/multiple run restriction also applies to the `-count` option as it is for a regular sampling run.

3.3.4 The Event Count File Format

When the `–count` option is specified, the sampling collector generates a file. This file contains the count information for all the events, selected across all the runs, in a session. Only one file will be generated per sampling session. The count file contains two types of output: informational and data.

- Each informational output always starts with `#sep:` keyword as the first argument ($1 Perl notation). The next keyword in the informational output represents the type of the information.
- Currently the supported informational output types are the following:
  - `version` -- version of the sampling collector
  - `header` -- format of the event count data
  - `date` – date and time when the session was initiated (may not be available for v1)
- The event count data follows after the informational output. The format of the data output should be specified in the informational output (mostly in the "header" keyword).

3.3.5 Fixed Counter Support

On the Intel(R) Atom(TM) processor, three fixed counters are implemented to count CPU_CLK_UNHALTED.CORE ("Clockticks"), INST_RETIRED.ANY ("Instructions Retired"), and CPU_CLK_UNHALTED.REF ("Referenced Cycles") respectively. When one or more of these three events are specified in the event configuration, the corresponding fixed counters are used. The two general counters are still available for counting other PMU events. Therefore, you can count up to five events in a single run: three fixed events and two general events.

In the following example, the sampling collector counts all five events in a single run:

```
sep -start -d 20 -ec "CPU_CLK_UNHALTED.CORE", "INST_RETIRED.ANY",
"CPU_CLK_UNHALTED.REF", "DATA_TLB_MISSES.DTLB_MISS",
"MEM_LOAD_RETIRED.L2_MISS" -out my_data
```

3.4 Usage Examples

This section provides several sample command lines demonstrating the usage of the sampling collector.

Table 3: Command Usage

Operation	Command line
Start data collection	`sep -start <"collector options">`
Pause data collection	`sep [-pause]`
Resume a paused data collection session	`sep [-resume]`
Stop data collection	`sep [-stop]`
get the list of events supported on the platform	`sep [-el | -event-list]`
Get help	`sep [-help | /?]`

Get version or build information	`sep [-version]`

4 Viewing Collected Data

To view the data collected on an MID Linux platform, another non-MID Linux platform supported by the VTune analyzer 9.1 is required (see the product Release Notes for the list of supported platforms). Install the VTune analyzer 9.1 on the non-MID Linux platform and select between using viewers from VTune analyzer command-line tool (`vtl`) or Eclipse* GUI tool (`vtlec`).

On the MID Linux machine, collect data using the command-line sampling collector. For example:

```
sep -start -d 10 -out my_data
```

This generates a sampling data file `my_data.tb5`. The next step is to copy the `tb5` file to the non-MID machine for importing.

4.1 Using vtl to View Sampling Data

To import and view the data with the `vtl` command-line tool, copy the `tb5` file to the non-MID Linux machine and use the following commands:

```
vtl import my_data.tb5          # imports the tb5 file into a new
                                  Activity

vtl show                        # shows list of imported Activities

vtl view -ar f1 -processes      # opens the process view in text
                                  mode for Activity f1

vtl view -ar f1 -gui            # shows results in GUI mode for
                                  Activity f1

man vtl                         # opens the manual page for vtl
```

4.2 Using vtlec to View Sampling Data

To import and view the data with the `vtlec` GUI tool, do the following:

1. Copy the `tb5` file to the non-MID Linux machine.

2. Start up `vtlec`.

3. Go to **File->Open File...** and specify the `tb5` file to import.
   This creates a new project icon in the **Tuning Browser** window in the Eclipse workspace.

4. Double-click the imported sampling results icon to view the `tb5` data.

NOTE: When drilling down to hotspots or sources in `vtl` or `vtlec`, you may need to copy additional files (for example, application binaries, libraries, source codes) to the non-MID machine.