AMD µProf User Guide

version 1.0

AMD Developer Tools Team

June 22, 2017

Contents

Welcome to AMD µProf's documentation!

Contents:

Introduction

AMD µProf is a comprehensive tool suite that allows developers to harness the benefits of AMD CPUs and APUs. This toolsuite is to help the developers to optimize their application running on AMD processors. It offers following functionalities:

- *AMDCpuProfiler*
- *AMDPowerProfiler*

AMD µProf is available as a command-line user interface application for Windows® operating system and Linux® operating system.

This document is intended for software developers. The chapter on CPU Profiling assumes an understanding of the concepts of threads and processes, as well as familiarity with CPU architecture.

System Requirements

Operating Systems

- Microsoft Windows 7 64-bit
- Microsoft Windows 8.1 64-bit
- Microsoft Windows 10 64-bit
- Ubuntu 15.04 64-bit & later
- RHEL 7.0 64-bit & later

Power Profiling

Supported platforms - Any of the AMD CPUs or APUs

Power Profiling

Supported platforms - Kaveri, Mullins, Temash or Carrizo APU

For detailed system requirements see the Release Notes.

Installation Instructions

Install AMD µProf using one of the following methods:

Windows

- Run AMDµProf-x.x.x.exe

Linux

- **Extract the AMD µProf tarball**
  - $ tar -xvzf AMDuProf_Linux_x64_x.x.x.tar.gz
- **Install the Power Profiler’s Linux driver manually with root credentials**
  - $ sudo <AMDµProf-install-dir>/AMDPowerProfilerDriverInstall.run

Support

Latest Version of This Document

For the latest version of this documentation, go to AMD µProf

AMDCpuProfiler

Introduction

AMDCpuProfiler is used for performance analysis and tuning of applications running on CPU. The CPU Profiler lets you identify main performance bottlenecks of the profiled application or the entire system.

Features

- **Supported Profile Types:**
    - **Time-Based Profile (TBP)** - To identify the “hot-spots” in the profiled applications. (Hot-spots are code areas that use significantly more time compared to other areas in the code.)
    - **Event-Based Profile (EBP)** - To identify CPU and memory related performance issues in the profiled applications.
    - **Instruction-Based Sampling (IBS)** - To record and count the instructions that trigger HW events, as well as calculate various metrics, such as data cache latency.
- **Supported Profile Modes:**
    - **Per-Process** - Profiles the launched process and it's children.
    - **System-Wide** - Profiles the entire system.
    - Attach to an aready running process.
    - **User Mode** and **Kernel Mode** Profiling.
- **Supported Languages**
    - C, C++ and Fortran applications
    - Java applications
    - CLR/.NET applications (only on Windows)
- Call Stack Sampling (CSS) for all profile modes.
- **Sample Aggregation:**
    - Process/Modules/Functions/Source and Instruction level aggregation.
    - Instruction Mix (IMIX)
- **Miscellaneous Features:**
    - Profiling C++ inline functions.
    - HW events counter multiplexing.
- **Debugging Info Formats Supported:**
    - Unmanaged Executables compiled by MS Visual Studio or GCC (under Linux or other Unix-like systems (like Cygwin and MinGW)).
    - Debug Information Formats - PDB, COFF, DWARF, STABS.
    - Managed Executables - JVMTI for Java applications and COR for .NET applications.
- **Supporting Virtualized Environments:**
    - Time-Based Profile (TBP) and Event-Based Profile (EBP) are supported in guest OS running on VMware Workstation 11.0 or later.
    - Time-Based Profile (TBP) and Event-Based Profile (EBP) are supported on Microsoft Hyper-V.
    - Time-Based Profile (TBP) is supported on Xen Project hypervisor.
    - Time-Based Profile (TBP) is supported on Linux KVM hypervisor.
- APIs to control CPU Profiling (i.e. to pause and resume profiling) from the target application to limit the profiling scope.

CPU Profile Key Concepts

This section explains various key concepts related to CPU Profiling. The AMD µProf offers CPU-Profiling through AMDCpuProfiler tool.

Profiling Concepts

Statistical Sampling

The AMDCpuProfiler follows a statistical sampling-based approach to gather the profile data periodically. It uses a variety of SW and HW resources available in AMD x86 based processor families. CPU Profiler uses the SW timer, HW Performance Monitor Counters (PMC), and HW IBS feature. The most time-consuming parts of a program have a larger number of samples; this is because they have a higher probability of being executed while samples are being taken by the CPU Profiler.

Sampling Interval

The time between the collection of every two samples is the Sampling Interval. For example, in TBP, if the time interval is 1 millisecond, then roughly 1,000 TBP samples are being collected every second for each processor core.

Hardware Sources

HW Performance Monitor Counters (PMC)

AMD's x86-based processors have Performance Monitor Counters (PMC) that let them monitor various micro-architectural events in a CPU core. The PMC counters are used in two modes:

- In counting mode, these counters are used to count the specific events that occur in a CPU core.
- In sampling mode, these counters are programmed to count a specific number of events;

once the count is reached the appropriate number of times (called sampling interval), an interrupt is triggered. During the interrupt handling, the CPU Profiler collects profile data.

The number of hardware performance event counters available in each processor is implementation-dependent (see the BIOS and Kernel Developer's Guide [BKDG] of the specific processor for the exact number of hardware performance counters). The operating system and/or BIOS can reserve one or more counters for internal use. Thus, the actual number of available hardware counters may be less than the number of hardware counters. The CPU Profiler uses all available counters for profiling.

Instruction-Based Sampling (IBS)

IBS is a code profiling mechanism that enables the processor to select a random instruction fetch or micro-Op after a programmed time interval has expired and record specific performance information about the operation.

An interrupt is generated when the operation is complete as specified by IBS Control MSR. An interrupt handler can then read the performance information that was logged for the operation.

The IBS mechanism is split into two parts:

- Instruction Fetch performance
- Instruction Execution Performance

Instruction fetch sampling provides information about instruction TLB and instruction cache behavior for fetched instructions.

Instruction execution sampling provides information about micro-Op execution behavior.

The data collected for instruction fetch performance is independent from the data collected for instruction execution performance. Support for the IBS feature is indicated by the Core::X86::Cpuid::FeatureExtIdEcx[IBS].

Instruction execution performance is profiled by tagging one micro-Op associated with an instruction. Instructions that decode to more than one micro-Op return different performance data depending upon which micro-Op associated with the instruction is tagged. These micro-Ops are associated with the RIP of the next instruction.

In this mode, the CPU Profiler uses the IBS HW supported by the AMD x86-based processor to observe the effect of instructions on the processor and on the memory subsystem. In IBS, HW events are linked with the instruction that caused them. Also, HW events are being used by the CPU Profiler to derive various metrics, such as data cache latency.

IBS is supported starting from the AMD processor family 10h.

Profile Types

Time-Based Profile (TBP)

In this profile mode, the profile data is periodically collected based on the specified timer interval. It is used to identify the hot-spots of the profiled applications.

Event-Based Profile (EBP)

In this mode, the CPU Profiler uses the PMCs to monitor the various micro-architectural events supported by the AMD x86-based processor. It helps to identify the CPU and memory related performance issues in profiled applications. AMDCpuProfiler provides a number of predefined EBP profile configurations. To analyze a particular aspect of the profiled application (or system), a specific set of relevant events are grouped and monitored together. The CPU Profiler provides a list of pre-defined event configurations, such as Assess Performance and Investigate Branching, etc. You can select any of these pre-define configurations to profile and analyze the runtime characteristics of your application. You also can create their custom configurations of events to profile.

This profile mode is supported on the various AMD processor models, such as Family 10h, Family 11h, Family 12h, Family 14h, Family 15h models 00h-0Fh, 10-1Fh, 30-3Fh and Family 16h models 00-0Fh.

In this profile mode, a delay called skid occurs between the time at which the sampling interrupt occurs and the time at which the sampled instruction address is collected. This skid distributes the samples in the neighborhood near the actual instruction that triggered a sampling interrupt. This produces an inaccurate distribution of samples and events are often attributed to the wrong instructions.

Instruction-Based Sampling (IBS)

In this mode, the CPU Profiler uses the IBS HW supported by the AMD x86-based processor to observe the effect of instructions on the processor and on the memory subsystem. In IBS, HW events are linked with the instruction that caused them. Also, HW events are being used by the CPU Profiler to derive various metrics, such as data cache latency.

Event-Counter Multiplexing

If the number of monitored PMC events is less than, or equal to, the number of available performance counters, then each event can be assigned to a counter, and each event can be monitored 100% of the time. In a single-profile measurement, if the number of monitored events is larger than the number of available counters, the CPU Profiler time-shares the available HW PMC counters. (This is called event counter multiplexing.) It helps monitor more events and decreases the actual number of samples for each event, thus reducing data accuracy. The CPU Profiler auto-scales the sample counts to compensate for this event counter multiplexing. For example, if an event is monitored 50% of the time, the CPU Profiler scales the number of event samples by factor of 2.

Command Line Interface

AMDCpuProfiler CLI utility will be used to collect and analyze the profile data. It can also be used from a batch file or a test script.

Usage:

```
> AMDCpuProfiler.exe command <options> [<application>] [<arguments>]
```

Following commands are supported:

`collect`	Run the given input application and collect cpu profile samples.
`report`	Process the given cpu profile data file and generate a cpu profile report in CSV format.

Following options are supported with `collect` command.

Option	Description

`--config` <collect config>	**Predefined data-collection configuration to be used to collect samples. Supported profile types are:** `tbp` : Time-based Sampling `assess` : Assess Performance `branch` : Investigate Branching `data_access` : Investigate Data Access `inst_access` : Investigate Instruction Access `l2_access` : Investigate L2 Cache Access `ibs` : Instruction-based Sampling

`-e`, `--event` <EVENT>	**Specify Timer, PMU or IBS event in the form of comma separated key=value pair. Supported keys are:**
	`event=<PMU-event-select>`
	`event=<timer | ibs-fetch | ibs-op>`
	`umask=<unit-mask>`
	`user=<0 | 1>`
	`os=<0 | 1>`
	`interval=<sampling-interval>`
	`ibsop-count-control=<0 | 1>`
	The PMU-event-select and unit-mask values can be in decimal or hex (prefixed with 0x). Default values are umask=0, user=1, os=1, interval=0, ibsop-count-control=0. For a PMC event, if the interval is not set or 0, then the event will be monitored in count mode. For timer, ibs-fetch and ibs-op events valid sampling interval is required. For timer, the interval is in milliseconds. If ibsop-count-control is 0, then count clock cycles otherwise count dispatched micro ops. Multiple occurrences of `-e` is allowed.

`-p`, `--pid` <PID,PID,..>	Profile existing processes (processes to attach to). Process IDs are separated by comma.
`-a`, `--system-wide`	System Wide Profile (SWP). If this flag is not set then the command line tool will profile only the launched application or the PIDs attached.
`-c`, `--cpu` <core-id,..>	Comma separated list of CPUs to profile. Ranges of CPUs also be specified with `-`, e.g. `0-3`.
`--call-graph <I:D :Scope:Fpo>`	**Enable Callstack Sampling. Specify the Unwind Interval and Unwind Depth values.** `user` : Collect only for user space code. `kernel` : Collect only for kernel space code. `all` : Collect for code executed in user and kernel space code. **Specify to collect missing frames due to omission of frame pointers by compiler.** `fpo` : Collect missing callstack frames. `nofpo` : Do not collect missing callstack frames. Option Scope and Fpo are only supported on Windows.
`-g`	**Enable callstack sampling with default Callstack collection sampling interval and Unwind Depth.** Unwind Interval : 1 Unwind Depth : 128 Scope : User FPO : No Option Scope and Fpo are only supported on Windows.
`-d`, `--duration` <n>	Profile duration ('n' in seconds).

`--affinity`	Core Affinity. Comma separated list of CPUs. Ranges of CPUs also be specified, e.g. `0-3`. Default affinity is all the available cores. In Per-Process profile, processor affinity is set for the launched application.
`--no-inherit`	Profile the children of the launched application (i.e. processes launched by the profiled application).
`-b`, `--terminate`	Terminate the launched application after profile data collection ends. Only the launched application process will be killed. Its children, if any, may continue to execute.
`--start-delay` <n>	Start Delay ('n' in seconds). Start profiling after the specified duration. If 'n' is 0, then wait indefinitely. Used for profile control. This option is expected to be used only when the launched application is instrumented to control the profile data collection using the enable or disable APIs defined in AMDProfileControl library, which is provided along with AMD µProf installer.
`-w`, `--working-dir` <dir>	Specify the working directory. Default will be the path of the launched application.
`-o`, `--output` <file name>	Base name of the output file. If this option is skipped, default path will be used. The default path will be `%Temp%\AMD-CpuProfile-<timestamp>.*` on Windows and `/tmp/AMD-CpuProfile-<timestamp>.*` on Linux.

Following options are supported with `report` command.

Option	Description
`-i`, `--input` <file name>	Input file name. Either the raw profile data file (.prd on Windows and .caperf on Linux) or the processed data file (.db) can be specified.
`-o`, `--output` <output dir>	Output directory in which the processed data file .db, .csv will be created. The default path will be `%Temp%\<base-name-of-input-file>.*` on Windows and `/tmp/<base-name-of-input-file>.*` on Linux.
`--summary`	Report only the overview of the profile.
`--config` <report config>	Predefined report configuration to be used while reporting the profile data. By default, all the raw data will be reported.
`--group-by` <section>	**Specify the report to be generated. Supported report options are:** `process` : Report process details. `module` : Report module details. `thread` : Report thread.
`--src`	Generate detailed function report with source statements.
`--disasm`	Generate detailed function report with assembly instructions.
`--imix`	Generate Instruction MIX report.
`--sort-by`	Specify the event index on which the reported profile data will be sorted. This event is also used to generate CallGraph section.
`--ignore-system-module`	Ignore samples from System Modules.
`--show-percentage`	Show Percentage.
`--src-path` <path1;..>	Source file directories. (Semicolon separated paths.)
`--symbol-path` <path1;..>	Debug Symbol paths. (Semicolon separated paths.)
`--symbol-server` <path1;..>	Symbol Server directories. (Semicolon separated paths.)

`--symbol-cache-dir` <path>	Path to store the symbols downloaded from the Symbol Servers.
`--cutoff` <n>	Cutoff to limit the number of process, threads, modules and functions to be reported. n is the minimum number of entries to be reported in various report sections. Default value is 10.

Following common options are supported with or without any command.

Option	Description
`--version`	Print version string.
`-h`, `--help`	Displays this help information.
`-v`, `--verbose` <n>	**Specify debug log messaging level. Valid values are 1 to 3.** 1 – INFO 2 – DEBUG 3 – EXTENSIVE

Examples

Print help:

```
> AMDCpuProfiler.exe -h
```

Print version string:

```
> AMDCpuProfiler.exe --version
```

Launch the application classic.exe and collect Time-based profile (TBP) samples:

```
> AMDCpuProfiler.exe collect -o c:\Temp\cpuprof-tbp classic.exe
```

Launch the application classic.exe and collect assess performance profile samples for the duration of 10 seconds:

```
> AMDCpuProfiler.exe collect --config assess -o c:\Temp\cpuprof-assess
-d 10 classic.exe
```

Launch the application classic.exe and collect Instruction Based Sampling (IBS) samples in System wide profile (SWP) mode:

```
> AMDCpuProfiler.exe collect --config ibs -a -o c:\Temp\cpuprof-ibs-swp classic.exe
```

Collect Time-based profile samples in System wide profile mode for the duration of 10 seconds:

```
> AMDCpuProfiler.exe collect -a -o c:\Temp\cpuprof-TBP-swp -d 10
```

Launch the application classic.exe and collect Time-based profile (TBP) samples. Also enable collecting callstack samples whenever the TBP samples are collected:

```
> AMDCpuProfiler.exe collect --config tbp -G -o c:\Temp\cpuprof-tbp classic.exe
```

Launch Classic.exe and collect samples for PMC events 0x76 and 0xc0:

```
> AMDCpuProfiler.exe collect -e event=0x76,interval=250000 -e
event=0xc0,user=1,os=0,interval=250000 -o c:\Temp\cpuprof-tbp classic.exe
```

Launch Classic.exe and collect samples for IBS OP with interval 50000:

```
> AMDCpuProfiler.exe collect -e event=ibs-fetch,interval=50000
-o c:\Temp\cpuprof-tbp classic.exe
```

Once the raw cpu profile data file is generated, AMDCpuProfiler report command can be used to generate CSV report from that raw data file:

```
> AMDCpuProfiler.exe report -i c:\Temp\cpuprof-tbp.prd -o c:\Temp\cpuprof-tbp-out
```

Generate report with Symbol Server paths:

```
> AMDCpuProfiler.exe report --symbol-path C:\AppSymbols;C:\DriverSymbols
--symbol-server http://msdl.microsoft.com/download/symbols
--cache-dir C:\symbols -i c:\Temp\cpuprof-tbp.prd -o c:\Temp\cpuprof-tbp-out
```

Limitations

AMDCpuProfiler expects the profiled application executable binaries must not be compressed or obfuscated by any software protector tools, e.g. VMProtect.

AMDPowerProfiler

Introduction

AMDPowerProfiler is a powerful tool to help analyzing the energy efficiency of systems based on AMD CPU, APUs and majority of the recent dGPU(discrete GPU).

Features

- **Report the following data:**
    - Estimated average energy consumed by CPU core and Socket.
    - Estimated average power consumed by APU and supported dGPU subcomponents.
    - Average frequency of the CPU cores and the internal GPU and supported dGPU.
    - Thermal trend of the CPU compute-units and the internal GPU.
    - Thermal trend of supported dGPU.
    - CPU cores P-States.
- A command-line tool to for data collection and dump to text/binary format.
- **Following hardware's are supported**
    - AMD APUs: Ryzen,Carrizo, Kaveri, Mullins, Temash, Stoney, Bristol
    - AMD dGPUs: Graphics IP 7 GPUs, Radeon and FirePro models.

Installing the Power Profiler Linux Driver

On Linux systems, the AMD µProf Debian and RPM packages perform the driver installation automatically. However, if you've downloaded the AMD µProf tar archive, you have to install the Power Profiler's Linux driver manually. This includes a simple step of running AMDPowerProfilerDriver.sh script with root credentials.

Example:

```
$ tar –xf AMDuProf_Linux_x64_x.x.x.tar.gz
$ cd AMDuProf_Linux_x64_x.x.x
$ sudo ./AMDPowerProfilerDriver.sh install
```

Installer will create a source tree for power profiler driver under `/usr/src/AMDPowerProfiler-<version number>`. All the source files required for module compilation is located in this directory are under MIT license.

To uninstall the driver run the following command:

```
$ cd <AMDµProf-install-dir>
$ sudo ./AMDPowerProfilerDriver.sh uninstall
```

Wider Linux power profiling support (DKMS)

On Linux machine Power Profiler driver can also be installed with Dynamic Kernel Module Support (DKMS) framework support. DKMS framework automatically upgrades the power profiler driver module whenever there is a change in the existing kernel. This saves user from manually upgrading the power profiler driver module. The DKMS package needs to be installed on target machines before running the installation steps mentioned in the above section. AMDPowerProfilerDriver.sh installer script will automatically takes care of DKMS related configuration if DKMS package is installed in the target machine.

Example (for Ubuntu system):

```
$ sudo apt-get install dkms
$ tar –xf AMDuProf_Linux_x64_x.x.x.tar.gz
$ cd AMDuProf_Linux_x64_x.x.x
$ sudo ./AMDPowerProfilerDriver.sh install
```

If the user upgrades the kernel version frequently it is recommended to use DKMS for installation.

Power Profiler’s Performance Counters

Category	Name	Description
Energy	RAPL Core energy	RAPL Package energy RAPL (Running Average Power Limit) MSR based energy counters for package and physical cores. Note: Available only on AMD Ryzen
Power	Total APU Power	Average APU Power for the sampling period, reported in Watts. This is an estimated consumption value which is calculated based on APU activity levels.
Power	CPU Compute Unit Power	Average CPU Compute Unit Power for the sampling period, reported in Watts. This is an estimated consumption value which is calculated based on APU activity levels.
Power	iGPU Power	Average Integrated-GPU Power for the sampling period, reported in Watts. This is an estimated consumption value which is calculated based on APU activity levels.
Power	PCIe-Controller	Average PCIe-Controller Power for the sampling period, reported in Watts. This is an Power estimated consumption value which is calculated based on APU activity levels. This value does not include the power consumed by PCIe devices connected to the PCIe bus.
Power	Memory-Controller	Average DDR Memory-Controller Power for the sampling period, reported in Watts. This Power is an estimated consumption value which is calculated based on APU activity levels. This value does not include the power consumed by the memory DIMMs.
Power	Display-Controller	Average Display-Controller Power for the sampling period, reported in Watts. This value Power refers to the APUs internal display controller which may be used in notebook and embedded configurations. This is an estimated consumption value which is calculated based on APU activity levels. This value does not include the power consumed by the display.
Power	Cumulative APU	The accumulated energy consumed by the APU throughout the profile session. Reported Power in Joules. Available only in the command line tool.
Power	Cumulative Compute	The accumulated energy consumed by the CPU Compute Unit throughout the profile Unit Power session. Reported in Joules. Available only in the command line tool.
Power	Cumulative iGPU Power	The accumulated energy consumed by the APU’s Internal GPU throughout the profile session. Reported in Joules. Note: Available only in the command line tool.
Power	dGPU power	Average Discrete-GPU Power for the sampling period, reported in Watts. This is an estimated consumption value which is calculated based on dGPU activity levels. The dGPU family name is prefixed with this counter name.
Frequency	CPU Core/Thread Average Frequency	Average CPU Core Frequency for the sampling period, reported in MHz. This is the Core Effective Frequency (CEF). The core can go into various P-States within the sampling period, each with its own frequency. The CEF is the average of the core frequencies over the sampling period.

Frequency	iGPU Average Frequency	Average Integrated-GPU Frequency for the sampling period, reported in MHz.
Frequency	dGPU Average Frequency	Average Discrete-GPU Frequency for the sampling period, reported in MHz. The dGPU family name is prefixed with this counter name.
Frequency	CPU Core/Thread Frequency Histogram	Histogram of CPU Core Effective Frequency (average frequency for the sampling period). Note: Available only in the command line tool.
Frequency	iGPU Frequency Histogram	Histogram of Internal-GPU Effective Frequency (average frequency for the sampling period). Note: Available only in the command line tool.
Temperature	CPU Compute-Unit Temperature	Measured CPU Compute Unit Average Temperature, reported in Celsius. The reported Measured value is normalized and scaled, relative to the specific processor's maximum operating temperature. This value can be used to indicate rise and decline of temperature.
Temperature	iGPU Measured	Measured Integrated-GPU Average Temperature, reported in Celsius. The reported value Temperature is normalized and scaled, relative to the specific processor's maximum operating temperature. This value can be used to indicate rise and decline of temperature.
Temperature	dGPU Measured Temperature	Measured Discrete-GPU Average Temperature, reported in Celsius. The reported value is normalized and scaled, relative to the specific processor's maximum operating temperature. This value can be used to indicate rise and decline of temperature. The dGPU family name is prefixed with this counter name.
DVFS	CPU P-State	CPU Core P-State at the time when sampling was performed.

Power Profiler Command Line Interface

AMDPowerProfiler provides a command line interface utility for users who prefer to use command interpreters like cmd.exe on Windows and bash on Linux. This CLI utility can be used to collect and analyze the profile data. It can also be used from a batch file or a test script.

Usage:

- Windows: `AmdPowerProfiler.exe <options>`
- Linux: `AmdPowerProfiler <options>`

Collect Options:

Short Option	Long Option	Details and Arguments

`-e`	`--event`	Collect counters for specified combination of device type and/or category type. **Supported device list:** `Socket` : Collect profile data from socket. `Die` : Collect profile data from die. `Core` : Collect profile data from core. `Thread` : Collect profile data from thread. **Supported category list:** `Power` : Collect all available power counters. `Frequency` : Collect all available frequency counters. `Temperature` : Collect all available temperature counters. `Voltage` : Collect all available voltage counters. `Current` : Collect all available current counters. `DVFS` : Collect all available dvfs counters. `Energy` : Collect all available energy counters. `CorrelatedPower` : Collect all available correlatedpower counters. `Cac` : Collect all available cac counters. `Controllers` : Collect all available controllers counters. Note: Multiple occurrences of `-e` is allowed.
n/a	`--process`	Enable process profiling.
n/a	`--module`	Enable module profiling.
n/a	`--histogram`	Collect histogram counters. Note : Allowed only with an occurrence of `-e` frequency.
n/a	`--cumulative`	Collect cumulative counters. Note : Allowed only with an occurrence of `-e` power.
`-t`	`--interval`	Sampling interval in milli seconds.The minimum value is 10ms.
`-d`	`--duration`	Profile duration in seconds.
`-c`	`--affinity`	Setting affinity to the launched target application
`-w`	`--working-dir`	Setting working directory for the launched target application
n/a	`--db`	Create database file for the profile report

Report Options:

Short Option	Long Option	Details and Arguments
`-f`	`--format`	Output file format txt or csv.Default file format is csv.
`-o`	`--output`	Output file

Common Options:

Short Option	Long Option	Details and Arguments
`-h`	`--help`	Displays this help information.
n/a	`--version`	Print version string.

Examples

1. **Collect all the power counter values for the duration of 10 seconds with sampling interval of 100 milliseconds:**

```
> AMDPowerProfiler.exe collect --event power --interval 100 --duration 10
```

2. **Collect all frequency counter values for 10 seconds, sampling them every 500 milliseconds and dumping the results to a csv file:**

```
> AMDPowerProfiler.exe collect --event frequency -o %Temp%\PowerOutput
--interval 500 --duration 10
```

3. **Collect all frequency counter values at core 0 to 3 for 10 seconds, sampling them every 500 milliseconds and dumping the results to a text file:**

```
> AMDPowerProfiler.exe collect --event core=0-3,frequency --output
%Temp%\PowerOutput --interval 500 -duration 10 --format txt
```

4. **Collect process power consumption values for 100 seconds, sampling them every 10 milliseconds and dumping the results to a csv file:**

```
> AMDPowerProfiler.exe collect --process --output %Temp%\PowerOutput
--interval 10 --duration 100
```

5. **Collect module power consumption values for 100 seconds, sampling them every 10 milliseconds and dumping the results to a csv file:**

```
> AMDPowerProfiler.exe collect --module --output %Temp%\PowerOutput
--interval 10 --duration 100
```

6. **Display help:**

```
> AMDPowerProfiler.exe -h
```

AMDTPowerProfileAPI Library

The AMDPowerProfile API library is useful to analyze the energy efficiency of systems based on AMD CPUs, APUs and dGPUs (Discrete GPU). These APIs provide interface to read the power, thermal and frequency characteristics of APU/dGPU and their subcomponents. These APIs are targeted for software developers who want to write their own application to sample the power counters based on their specific use case.

AMDTPowerProfileAPI shared library has dependencies on AMDTBaseTools and AMDTOSWrappers shared libraries, so the corresponding .DLL (on Windows system) and .SO (on Linux system) should be added.

To build and execute a test application (test.cpp) following steps should be performed on Linux machine.

1. **Assuming test.cpp is located at** `/home/<user-dir>/samples`

```
$ cd /home/<user-dir>/samples
```

2. **Set LD_LIBRARY_PATH**

```
$ export LD_LIBRARY_PATH=<AMDµProf-install-dir>
```

3. **Compile application code**

```
$ g++ test.cpp -I<AMDµProf-install-dir>/SDK/AMDTPowerProfile/inc
-L<AMDµProf-install-dir>/SDK/AMDTPowerProfile/bin/x86_64
-lAMDTPowerProfileAPI -L<AMDµProf-install-dir> -lAMDTOSWrappers
-lAMDTBaseTools -o test
```

4. **Execute**

```
$ ./test
```

Note: When using the static AMDPowerProfile library for the Power Profiler API on Linux, the user must build his application with `-Wl,--whole-archive -lpthread -Wl,--no-whole-archive`. Otherwise not all the symbols from pthread library will be linked, since most of them are WEAK symbols. Failing to use these flags will lead to a crash. (1040)

Limitations

- Multiple instance of AMDPowerProfiler cannot be run simultaneously.
- ICELAND discrete GPU(Topaz-XT, Topaz PRO, Topaz XTL, Topaz LE) series is not support in 2.0 release.

- Please make sure you have latest Radeon driver installed before running power profiler. Newer version of discrete GPU may go to sleep (low power) state frequently if there is no activity in that GPU. In that case, power profiler may emits a warning AMDT_WARN_SMU_DISABLED. Counters may not be accessible during this state. It is advisable to bring discrete GPU to active state by running some openCL or openGL application, then run power profiling on that GPU.
- Process profiling is supported only with command line tool. If PMC (Performance Monitoring Counters) counters are not accessible and unable to calculate the IPC load, then compute unit power is distribute equally to each core. In that case power distribution to each process may not be accurate.
- Module profiling is supported only with command line tool. If PMC (Performance Monitoring Counters) counters are not accessible and unable to calculate the IPC load, then compute unit power is distribute equally to each core. In that case power distribution to each process may not be accurate. Modules which are started after the process run are not considered in this release.

SDKs

AMD µProf provides following development librarys for the developers:

AMDPowerProfileAPI

AMDPowerProfileApi library provides APIs to configure, collect and report the supported power profiling counters on various AMD platforms.

Refer <AMDµProf-install-dir>/Help/AMDPowerProfilerAPI.pdf for API details.

AMDProfileControl APIs

AMDProfilerControl APIs allow user to limit the profiling scope to a specific portion of the code within the target application. Usually, when the profiling done, it captures the samples for the complete application, i.e. start of execution till end of the application execution. The control APIs can be used to enable the profiler only for a specific part of application, e.g. a CPU intensive loop, a hot function, etc. The target application need to be recompiled after adding the control APIs within the application.

The control APIs:

To Pause CPU profiling, call one of the below two APIs.

// Set mode to AMDT_CPU_PROFILING

```
int amdtStopProfiling(amdtProfilingControlMode);
```

```
int amdtStopProfilingEx(void);
```

To resume CPU profiling, call one of the below two APIs.

// Set mode to AMDT_CPU_PROFILING

```
int amdtResumeProfiling(amdtProfilingControlMode);
```

```
int amdtResumeProfilingEx(void);
```

CPU Profiler only profiles the code within each Resume, Stop APIs pair. Refer “CPU Profiler Tutorial” on how to use these APIs, compile your target application and profile only the desired part of code.