# CONTENTS

## 1 Introduction
1.1 System Requirements .................................................. 1
1.2 Installation ............................................................... 1
1.3 Installing the Power Profiler Linux Driver ......................... 2
1.4 Wider Linux Power Profiler Support (DKMS) .......................... 2
1.5 Installing Remote Agent ................................................. 3
1.6 Support ................................................................. 3

## 2 Getting started with AMDuProf GUI
2.1 Welcome Page ................................................................ 4
2.2 Remote Profiling ............................................................ 5

## 3 CPU Profiler
3.1 Introduction ................................................................. 7
3.2 Key Concepts ............................................................... 8
3.3 Command Line Interface (CLI) ........................................... 10
3.4 Graphical User Interface (GUI) .......................................... 18
3.5 Limitations ............................................................... 35

## 4 Power Profiler
4.1 Introduction ................................................................. 36
4.2 Key Concepts ............................................................... 36
4.3 Command Line Interface (CLI) .......................................... 37
4.4 Graphical User Interface (GUI) .......................................... 41
4.5 Limitations ............................................................... 46

## 5 Performance Counter Monitor (PCM) - Linux Only
5.1 Introduction ................................................................. 47
5.2 Command Line Interface (CLI) .......................................... 47

## 6 SDKs
6.1 AMDPowerProfileAPILibrary ........................................... 49
6.2 AMDProfileControlAPIs ................................................. 59

## 7 Reference
7.1 CPU Performance Metrics ............................................... 61
7.2 Power Metrics .............................................................. 67
7.3 AMD Processor events reference ...................................... 69
AMD uProf is a comprehensive tool suite that allows developers to harness the benefits of AMD CPUs and APUs. This tool suite is to help the developers to optimize their application running on AMD processors. It offers following functionalities:

- **CPU Profiler**
- **Power Profiler**
- **Performance Counter Monitor (Linux only)**

AMD uProf is available as Graphical User Interface and Command Line Interface tools for Windows® operating system and Linux® operating system.

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

### 1.1 System Requirements

#### 1.1.1 Operating Systems

- Microsoft Windows 7 64-bit
- Microsoft Windows 10 64-bit
- Microsoft Windows Server 2016 64-bit
- Ubuntu 16.04 64-bit & later
- RHEL 7.0 64-bit & later

#### 1.1.2 CPU Profiling

- Supported platforms - AMD CPUs or APUs, and Intel CPUs

#### 1.1.3 Power Profiling

- Supported platforms - Kaveri, Mullins, Temash, Carrizo, Ryzen, Threadripper, EPYC
- Supported AMD dGPUs - Graphics IP 7 GPUs, AMD Radeon 500 Series and FirePro models

For detailed system requirements see the Release Notes.

### 1.2 Installation

Install AMD uProf using one of the following methods.
1.2.1 On Windows

Run AMDuProf-x.y.z.exe installer.

1.2.2 On Linux (using .tar.gz binary)

Just extract the tar.gz binary.

$ tar -xvzf AMDuProf_Linux_x64_x.y.z.tar.gz

To install the Power Profiler Linux driver, refer the next section.

1.2.3 On RHEL (using .rpm installer)

Install using rpm or yum command.

$ sudo rpm --install amduprof-x.y-z.x86_64.rpm
OR
$ sudo yum install amduprof-x.y-z.x86_64.rpm

1.2.4 On Ubuntu (using .deb installer)

Install using dpkg command.

$ sudo dpkg --install amduprof_x.y-z_amd64.deb

1.3 Installing the Power Profiler Linux Driver

On Linux systems, gcc and make software packages are prerequisites for installing Power Profiler’s Linux driver. If you don’t have these packages, they can be installed using sudo yum install gcc make (on RPM systems) and sudo apt install build-essential (on Debian systems).

AMD uProf Debian and RPM packages perform the driver installation automatically. However, if you’ve downloaded the AMD uProf 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.y.z.tar.gz
$ cd AMDuProf_linux_x64_x.y.z/bin
$ sudo ./AMDPowerProfilerDriver.sh install

Installer will create a source tree for power profiler driver under /usr/src/AMDPowerProfilerDriver.sh install. 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 <AMDuProf-install-dir>/bin
$ sudo ./AMDPowerProfilerDriver.sh uninstall

1.4 Wider Linux Power Profiler 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 take care of DKMS related configuration if DKMS package is installed in the target machine.

**Example (for Ubuntu system):**

```bash
$ sudo apt-get install dkms
$ tar -xf AMDuProf_Linux_x64_x.y.z.tar.gz
$ cd AMDuProf_Linux_x64_x.y.z/bin
$ sudo ./AMDPowerProfilerDriver.sh install
```

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

### 1.5 Installing Remote Agent

The uProf Remote Agent is shipped with the AMDuProf installer, and is installed by default when installing AMDuProf. You can also choose to install only AMDuProf Remote Agent when using the Windows installer.

### 1.6 Support

Visit the following sites for bug reports, support and feature requests.

- AMD uProf Release page
- AMD Developer Community
GETTING STARTED WITH AMDUPROF GUI

AMDuProf GUI provides a visual way to profile and analyze performance data.

2.1 Welcome Page

On opening the GUI, you will be greeted with the welcome screen. This screen contains various quick links and hyper-links to jump start workflows. It will look like this:

The various sections are:

1. Create a new profile? link lets you generate a new profile with requisite options. Open existing profile config? link lets you browse through various saved profile configurations and choose anyone. Connect to Remote Machine? link lets you connect to the remote target system.

2. Once you have profiled the application or imported a profile database, you will see the last 5 opened databases and a link to see the complete list as well in Recently Opened Profiles section.

3. The Quick Links section contains two entries which lets you to start profiles with minimal configuration. The first one will start a system-wide time-based profiling until stopped by user and then display the collected data. The second option takes to a section where numerous counters can be selected and will present a live view of the data through graphs.

4. The AMD Developer Blog section provides useful links for the developers.
2.2 Remote Profiling

AMDuProf provides remote profiling capabilities to perform CPU and Power profiling of applications that run on a remote system. This is useful for working with headless server units.

2.2.1 Running Remote Agent

The uProf remote agent **AMDRemoteAgent** runs on the remote system, and allows AMDuProf clients installed on other machines to connect to that remote machine. Once the connection is established, client can execute CPU and Power profiling sessions of applications running on that remote system.

When remote agent AMDRemoteAgent.exe is launched, it will output to the console a message in the following format:

```
c:\Program Files\AMD\AMDuProf\bin>AMDRemoteAgent.exe
Local connection: IP: 192.168.0.100, port 27016
Waiting for a remote connection...
```

2.2.2 Connecting AMDuProf client to remote system

AMDuProf GUI can be configured to connect to a remote system by clicking **Connect to a Remote machine?** link in the **Welcome** section. This will take you to the remote profile configuration page. The IP address and port needs to be configured. The port in which the remote agent is listening on the remote system should be specified. Clicking “Connect” button will initiate the connection to the remote platform.

Once the connection is established, the title bar of AMDuProf GUI will show the remote system’s IP address. All the settings, history and saved configurations are with respect to the remote system. Any profile type can be configured to profile on remote system.
User can switch back to the local profiling after disconnecting the remote connection.
3.1 Introduction

AMD uProf’s CPU Profiler is used for performance analysis and tuning of applications running on CPU. The CPU Profiler lets you identify top performance bottlenecks of the profiled application or the entire system.

3.1.1 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 Targets**
  - **Application** - Profiles the launched process and it’s children.
  - **System** - Profiles the entire system.
  - **Process** - Profile an already running process.

- **User 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 types.**

- **Sample Aggregation**
  - Process, Thread, Load Module, Function, 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) is 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.

3.2 Key Concepts

This section explains various key concepts related to CPU Profiling.

The uProf CPU Profiler follows a 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.

3.2.1 Hardware Sources

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.

L3 Cache Performance Monitor Counters (L3)

A Core Complex (CCX) is a group of CPU cores which share L3 cache resources. All the cores in a CCX share a single L3 cache. In family 17, 8MB of L3 cache shared across all cores within the CCX. Family 17 processors support L3PMCs to monitor the performance of L3 resources. Refer family 17 PPR for more details.

Data Fabric Performance Monitor Counters (DF)

Family 17 processors support DFPMCs to monitor the performance of Data Fabric resources. Refer family 17 PPR for more details.

3.2.2 Profiling Concepts

Sampling

Sampling profilers work based on the logic that the part of a program that consumes most of the time (or that triggers the most occurrence of the sampling event) 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.

The meaning of sampling interval depends on the resource used as the sampling event.

- SW timer - the sampling interval is in milliseconds.
- PMC events - the sampling interval is the number of occurrences of that sampling event
- IBS - the number of processed instructions after which it will be tagged.

Smaller sampling interval increases the number of samples collected and as well the data collection overhead. Since profile data is collected on the same system in which the workload is running, more frequent sampling increases the intrusiveness of profiling. Very small sampling interval also can cause system instability.

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.

3.2.3 Profile Types

Profile types are classified based on the HW or SW sampling events used to collect the profile data.

Time-Based Profile (TBP)
In this profile, the profile data is periodically collected based on the specified SW timer interval. It is used to identify the hot-spots of the profiled applications.

**Event-Based Profile (EBP)**

In this profile, 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. The CPU Profiler 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 predefined event configurations, such as Assess Performance and Investigate Branching, etc. You can select any of these predefined configurations to profile and analyze the runtime characteristics of your application. You also can create their custom configurations of events to profile.

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 profile, 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.

**Custom Profile**

This profile allows a combination of HW PMC events, SW timer, and IBS sampling events.

### 3.2.4 Performance Metrics

Following section describes some of the important CPU profiling metrics.

- **CPU Performance Metrics**

### 3.3 Command Line Interface (CLI)

AMD uProf’s CPU Profiler provides a command line interface utility `AMDuProfCLI` to collect and analyze the profile data. It can also be used from a batch file or a test script.

#### 3.3.1 Options

**Usage:**

Run the given input application, collect CPU profile samples and generate raw data files:

```
AMDuProfCLI collect <collect-options> [application] [app-args]
```

- `<collect-options>`: Check TABLE-1 below for complete list of collect command options.
- `<application>`: Path to the application to be profiled.
- `<app-args>`: Arguments (if any) to be passed to the application to be profiled.

Process the given CPU profile raw data file and generate report in CSV format:

```
AMDuProfCLI report <report-options>
```

- `<report-options>`: Check TABLE-2 below for complete list of report command options.
Displays generic information about system, topology etc.:

```
AMDuProfCLI info <info-options>
```

* `<info-options>`: Check TABLE-3 below for complete list of info command options.

Displays the version of AMD uProf tool:

```
AMDuProfCLI --version
```

Displays the help details on console/terminal:

```
AMDuProfCLI -h|--help
```

Opens this HTML help page in a web browser:

```
AMDuProfCLI help
```

Table 3.1: TABLE-1: ‘collect’ command options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--help</td>
</tr>
<tr>
<td>--list &lt;type&gt;</td>
<td>Lists the supported items for the following types:</td>
</tr>
<tr>
<td></td>
<td><code>collect-configs</code>: Predefined profile configurations that can be used with --config option.</td>
</tr>
<tr>
<td></td>
<td><code>pmu-events</code>: Raw PMU events that can be used with --event option.</td>
</tr>
<tr>
<td>--config &lt;collect-config&gt;</td>
<td>Predefined data-collection configuration to be used to collect samples. Use the command</td>
</tr>
</tbody>
</table>

Continued on next page
Table 3.1 – continued from previous page

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-e</td>
<td><strong>--event</strong> &lt;EVENT&gt; Specify Timer, PMU or IBS event with arguments in the form of comma separated key=value pairs. Supported keys are: event=&lt;timer</td>
</tr>
<tr>
<td>Use command collect --list pmu-events for the list of supported PMU-events. Details about the arguments: umask - Applicable to PMU events. It can be in decimal or hexadecimal. user, os - Applicable to PMU events. interval - Applicable to all events. For timer, the interval is in milliseconds. For PMU 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. ibsop-count-control - Applicable only to ibs-op event. When set to 0, count clock cycles, otherwise count dispatched micro ops. slicemask, threadmask - Applicable only to L3 PMU events. When these arguments are not passed, then the default values are: umask=0, user=1, os=1, interval=0, ibsop-count-control=0, slicemask=0xF, threadmask=0xFF Multiple occurrences of <strong>--event</strong> (e) are allowed. <strong>NOTE</strong>: L3 PMU events are supported only on Windows and only on Family 0x17 processors.</td>
<td></td>
</tr>
<tr>
<td>-p</td>
<td><strong>--pid</strong> &lt;PID,PID,..&gt; Profile existing processes (processes to attach to). Process IDs are separated by comma.</td>
</tr>
<tr>
<td>-a</td>
<td><strong>--system-wide</strong> System Wide Profile (SWP). If this flag is not set then the command line tool will profile only the launched application or the Process IDs attached with <strong>-p</strong> option.</td>
</tr>
<tr>
<td>-c</td>
<td><strong>--cpu</strong> &lt;core-id,..&gt; Comma separated list of CPUs to profile. Ranges of CPUs also be specified with -, e.g. 0-3. Use info --cpu-topology command to get list of available core-ids. <strong>NOTE</strong>: On Windows, the selected cores should belong to only one processor group, e.g. 0-63, 64-127 and so on.</td>
</tr>
<tr>
<td>--call-graph</td>
<td><strong>&lt;I:D:S:F&gt;</strong> [On Windows] Enable callstack Sampling. Specify the Unwind Interval (I) in milliseconds and Unwind Depth (D) value. Specify the Scope (S) by choosing one of the following: 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 Frame Pointer Omission (F) by compiler: fpo: Collect missing callstack frames. nofpo: Ignore missing callstack frames.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 3.1 – continued from previous page

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| --call-graph <F:N> | [On Linux] Enable Callstack sampling. Specify F to collect/ignore missing frames due to omission of frame pointers by compiler:  
  fpo - Collect missing callstack frames.  
  nofp0 - Ignore missing callstack frames.  
  When F = fpo, N specifies the max stack-size in bytes to collect per sample collection.  
  Valid range to stack size: 16 - 8192. If N is not multiple of 8, then it is aligned down to the nearest value multiple of 8. The default value is 1024 bytes.  
  NOTE: Passing a large N value will generate a very large raw data file.  
  When F = nofp0, the value for N is ignored, hence no need to pass it. |
| -g         | [On Windows] Same as passing --call-graph 1:128:user:nofpo  
[On Linux] Same as passing --call-graph nofpo |
| -d|--duration <n> | Profile only for the specified duration n in seconds. |
| --affinity <core-id,..> | Set the core affinity of the launched application to be profiled. Comma separated list of core-ids. Ranges of core-ids also be specified, e.g. 0-3. Default affinity is all the available cores. |
| --no-inherit | Do not 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. When n is 0, it has no impact. |
| --start-paused | Profiling paused indefinitely. The target application resumes the profiling using the profile control APIs. This option is expected to be used only when the launched application is instrumented to control the profile data collection using the resume and pause APIs defined in AMDProfileControl library, which is provided along with AMD uProf installer. |
| --w|--working-dir <dir> | Specify the working directory. Default will be the directory of the launch 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. |
| -v|--verbose <n> | Specify debug log messaging level. Valid values are:  
  1: INFO  
  2: DEBUG  
  3: EXTENSIVE |

Table 3.2: TABLE-2: ‘report’ command options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--help</td>
</tr>
<tr>
<td>--list view-configs</td>
<td>List of the supported report view configurations that can be used with --view option.</td>
</tr>
<tr>
<td>-i</td>
<td>--input &lt;file name&gt;</td>
</tr>
<tr>
<td>-o</td>
<td>--output &lt;output dir&gt;</td>
</tr>
<tr>
<td>--summary</td>
<td>Report only the overview of the profile. This is set by default.</td>
</tr>
</tbody>
</table>
Table 3.2 – continued from previous page

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--group-by &lt;section&gt;</td>
<td>Specify the report to be generated. Supported report options are:</td>
</tr>
<tr>
<td></td>
<td>process: Report process details.</td>
</tr>
<tr>
<td></td>
<td>module: Report module details.</td>
</tr>
<tr>
<td></td>
<td>thread: Report thread details.</td>
</tr>
<tr>
<td></td>
<td>Default is set to group-by process.</td>
</tr>
<tr>
<td>--cutoff &lt;n&gt;</td>
<td>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.</td>
</tr>
<tr>
<td>--view &lt;view-config&gt;</td>
<td>Report only the events present in the given view file. Use the command report --list view-configs to get the list of supported view-configs.</td>
</tr>
<tr>
<td>--src</td>
<td>Generate detailed function report with source statements.</td>
</tr>
<tr>
<td>--src-path &lt;path1;..&gt;</td>
<td>Source file directories. (Semicolon separated paths.)</td>
</tr>
<tr>
<td>--disasm</td>
<td>Generate detailed function report with assembly instructions.</td>
</tr>
<tr>
<td>--sort-by &lt;event-index&gt;</td>
<td>Specify the (0-based) event index on which the reported profile data will be sorted. This event is also used to generate CallGraph section and IMIX section. By default the first event (i.e. event index 0) is selected.</td>
</tr>
<tr>
<td>--imix</td>
<td>Generate Instruction MIX report.</td>
</tr>
<tr>
<td>--ignore-system-module</td>
<td>Ignore samples from system modules.</td>
</tr>
<tr>
<td>--show-percentage</td>
<td>Show percentage of samples, instead of actual samples.</td>
</tr>
<tr>
<td>--symbol-path &lt;path1;..&gt;</td>
<td>Debug Symbol paths. (Semicolon separated paths.)</td>
</tr>
<tr>
<td>--symbol-server &lt;path1;..&gt;</td>
<td>[On Windows] Symbol Server directories. (Semicolon separated paths.)</td>
</tr>
<tr>
<td>--symbol-cache-dir &lt;path&gt;</td>
<td>[On Windows] Path to store the symbol files downloaded from the Symbol Servers.</td>
</tr>
<tr>
<td>-v</td>
<td>--verbose &lt;n&gt;</td>
</tr>
<tr>
<td></td>
<td>1: INFO</td>
</tr>
<tr>
<td></td>
<td>2: DEBUG</td>
</tr>
<tr>
<td></td>
<td>3: EXTENSIVE</td>
</tr>
</tbody>
</table>

Table 3.3: TABLE-3: ‘info’ command options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--system</td>
<td>Displays processor information of this system.</td>
</tr>
<tr>
<td>--cpu-topology</td>
<td>Displays CPU topology information of this system.</td>
</tr>
<tr>
<td>--disasm &lt;binary-path&gt;</td>
<td>Disassembles the given binary file.</td>
</tr>
</tbody>
</table>

3.3.2 Examples (On Windows)

Collect Command

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

> AMDuProfCLI.exe collect -o c:\Temp\cpuprof-tbp AMDTClassicMatMul.exe

Launch AMDTClassicMatMul.exe and do ‘Assess Performance’ profile for 10 seconds:

> AMDuProfCLI.exe collect --config assess -o c:\Temp\cpuprof-assess -d 10 AMDTClassicMatMul.exe

Launch AMDTClassicMatMul.exe and collect ‘IBS’ samples in SWP mode:

> AMDuProfCLI.exe collect --config ibs -a -o c:\Temp\cpuprof-ibs-swp AMDTClassicMatMul.exe
Collect ‘TBP’ samples in SWP mode for 10 seconds:

> AMDuProfCLI.exe collect -a -o c:\Temp\cpuprof-tbp-swp -d 10

Launch AMDTClassicMatMul.exe and collect ‘TBP’ with Callstack sampling:

> AMDuProfCLI.exe collect --config tbp -g -o c:\Temp\cpuprof-tbp AMDTClassicMatMul.exe

Launch AMDTClassicMatMul.exe and collect ‘TBP’ with callstack sampling (unwind FPO optimized stack):

> AMDuProfCLI.exe collect --config tbp --call-graph 1:64:user:fpo -o c:\Temp\cpuprof-tbp AMDTClassicMatMul.exe

Launch AMDTClassicMatMul.exe and collect samples for PMCx076 and PMCx0C0:

> AMDuProfCLI.exe collect -e event=pmcx76,interval=250000 -e event=pmcx0C,user=1,os=0,interval=250000 -o c:\Temp\cpuprof-tbp AMDTClassicMatMul.exe

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

> AMDuProfCLI.exe collect -e event=ibs-op,interval=50000 -o c:\Temp\cpuprof-tbp AMDTClassicMatMul.exe

Collect L3 samples for event L3PMCx01 in SWP mode:

> AMDuProfCLI.exe collect -e event=timer,interval=1 -e event=L3pmcx01,umask=0x80,slicemask=0xF,threadmask=0xFF -a -d 10 -o c:\Temp\cpuprof-l3

Report Command

Generate report from the raw datafile:

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

Generate IMIX report from the raw datafile:

> AMDuProfCLI.exe report --imix -i c:\Temp\cpuprof-tbp.prd -o c:\Temp\cpuprof-tbp-out

Generate report with Symbol Server paths:

> AMDuProfCLI.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

Others

Open this HTML help in a web browser:

> AMDuProfCLI.exe help

Print help in the console/terminal:

> AMDuProfCLI.exe --help

Print uProf version:

> AMDuProfCLI.exe --version

Print system details:

> AMDuProfCLI.exe info --system

Print CPU topology details:
To disassemble AMDTClassicMatMul.exe into classic-disasm.txt file:
> AMDuProfCLI.exe info --disasm AMDTClassicMatMul.exe > classic-disasm.txt

3.3.3 Examples (On Linux)

Collect Command

Launch the application AMDTClassicMatMul-bin and collect Time-based profile (TBP) samples:
$ ./AMDuProfCLI collect -o /tmp/cpuprof-tbp AMDTClassicMatMul-bin

Launch AMDTClassicMatMul-bin and do ‘Assess Performance’ profile for 10 seconds:
$ ./AMDuProfCLI collect --config assess -o /tmp/cpuprof-assess -d 10 AMDTClassicMatMul-bin

Launch AMDTClassicMatMul-bin and collect ‘IBS’ samples in SWP mode:
$ ./AMDuProfCLI collect --config ibs -a -o /tmp/cpuprof-ibs-swp AMDTClassicMatMul-bin

Collect ‘TBP’ samples in SWP mode for 10 seconds:
$ ./AMDuProfCLI collect -a -o /tmp/cpuprof-tbp-swp -d 10

Launch AMDTClassicMatMul-bin and collect ‘TBP’ with Callstack sampling:
$ ./AMDuProfCLI collect --config tbp -g -o /tmp/cpuprof-tbp AMDTClassicMatMul-bin

Launch AMDTClassicMatMul-bin and collect ‘TBP’ with callstack sampling (unwind FPO optimized stack):
$ ./AMDuProfCLI collect --config tbp --call-graph fpo:512 -o /tmp/cpuprof-tbp AMDTClassicMatMul-bin

Launch AMDTClassicMatMul-bin and collect samples for PMCx076 and PMCx0C0:
$ ./AMDuProfCLI collect -e event=pmcx76,interval=250000 -e event=pmcx0,user=1,os=0,interval=250000 -o /tmp/cpuprof-tbp AMDTClassicMatMul-bin

Launch AMDTClassicMatMul-bin and collect samples for IBS OP with interval 50000:
$ ./AMDuProfCLI collect -e event=ibs-op,interval=50000 -o /tmp/cpuprof-tbp AMDTClassicMatMul-bin

Report Command

Generate report from the raw datafile:
$ ./AMDuProfCLI report -i /tmp/cpuprof-tbp.caperf -o /tmp/cpuprof-tbp-out

Generate IMIX report from the raw datafile:
$ ./AMDuProfCLI report --imix -i /tmp/cpuprof-tbp.caperf -o /tmp/cpuprof-tbp-out
3.3.4 How to Start Profiling

To profile and to analyze the performance of a native (C/C++) application, you need to follow 3 easy steps which can be summarized as:

1. Prepare the application.
2. Collect the samples for the application using uProf.
3. Generate the report in readable format for analysis.

Let’s look at the steps in detail.

STEP-1: (Prepare the application)

The uProf CPU Profiler uses the debug information generated by the compiler to show the correct function names in various analysis views and to attribute the collected samples to source statements in Source View. Otherwise, the results of the CPU Profiler would be less descriptive, displaying only the assembly code.

Generate Debug Information on Windows

When using Microsoft Visual C++ to compile the application in release mode, set the following options before compiling the application to ensure that the debug information is generated and saved in a program database file(with a .pdb extension). To set the compiler option to generate the debug information for a Win32 application in release mode:

1. Right click on the project and select ‘Properties’ menu item.
2. In the ‘Configuration’ list, select Active(Release).
3. In the ‘Platform’ list, select Active(Win32) or Active(x64).
4. In the project pane, expand the ‘Configuration Properties’ item, then expand the ‘C/C++’ item. Select ‘General’.
5. In the work pane, select ‘Debug Information Format’, and from the drop-down list select ‘Program Database (/Zi)’ or ‘Program Database for Edit & Continue (/ZI)’.
6. In the project pane, expand the ‘Linker’ item; then select the ‘Debugging’ item.
7. In the ‘Generate Debug Info’ list, select (/DEBUG).
Generate Debug Information on Linux

The application must be compiled with the `-g` option to enable the compiler to generate debug information. Modify either the Makefile or the respective build scripts accordingly.

STEP-2 (Collect the samples)

Select the suitable collect-config type. Refer TABLE-1 above for more details on collect-config types. Use collect command to profile the application. Refer ‘CLI Options’ and ‘Examples’ sections above. Once profiling is completed, uProf will generate raw data files (.prd on Windows, .caperf on Linux and few other miscellaneous files.)

STEP-3 (Generate the reports)

Use report command to generate the report (in CSV format) from the raw data file generated in step 2. Refer ‘CLI Options’ and ‘Examples’ sections above.

3.3.5 How To Profile Java Application

AMDuProfCLI supports Java application profiling running on JVM. To support this, it uses JVM Tool Interface (JVMTI).

AMDuProfCLI provides JVMTI Agent libraries: AMDJvmtiAgent.dll (on Windows), libAMDJvmtiAgent.so (on Linux). The JVMTI expects this JvmtiAgent need to be attached to the target JVM before starting the profiling. Only after attaching the JvmtiAgent, AMDuProfCLI would be able to collect the profile data. In this release, AMDuProfCLI can’t attach JvmtiAgent dynamically to an already running JVM. Hence any JVM process profiled by attach-process mechanism, AMDuProfCLI can’t capture any class information, unless the JvmtiAgent is attached to the JVM. As long as the Java application is launched by AMDuProfCLI, you don’t need to worry about JvmtiAgent, AMDuProfCLI would take care all these things in the background.

To profile a Java application, you may use command similar to the following sample command:

```
AMDuProfCLI collect --config tbp -w <java-app-dir>
<path-to-java.exe> <java-app-main>
```

To generate report, you may need to pass source file path:

```
AMDuProfCLI report --src-path <path-to-java-app-source-dir> -i
<raw-data-file>
```

If there is a scenario, where you need to attach to an already running JVM, you need to attach the uProf JvmtiAgent manually to the JVM while running the Java application using `-agentpath` option. Later AMDuProfCLI can attach to the JVM PID to collect profile data.

For a 64-bit JVM on Linux:

```
$ java
-agentpath:path-to-uProf/bin/ProfileAgents/x64/libAMDJvmtiAgent.so
<java-app-launch-options>
```

For a 64-bit JVM on Windows:

```
> java -agentpath:C:\Program Files\AMD\AMDuProf\bin\ProfileAgents\x64\AMDJvmtiAgent.dll
<java-app-launch-options>
```

Keep a note of the process id (PID) of the above JVM instance. Then launch AMDuProfCLI with the usual options and `--pid <jvm-pid>`.  

3.4 Graphical User Interface (GUI)

AMDuProf GUI provides a visual way to profile an application or the system and analyze the performance data.
3.4.1 Starting a CPU Profile

In order to start a profile, either click the PROFILE page at the top navigation bar or Create a new profile? hyperlink. This will navigate to the Start Profiling section of PROFILE page:
Selecting Profile Target

You can see Select Profile Target fragment in the Start Profiling section. In this fragment, different types of profile target can be selected from the Select Profile Target drop down list. Following profile target options are available:

Profiling an Application

Application: Select this target when you want to launch an application and profile it (or launch and do a system-wide profile). The only compulsory option is a valid path to the executable. (By default, the path to the executable becomes the working directory unless you specify a path). Once provided, it will look like this:

![Profiling an Application](image)

Profiling the System

System: Select this if you do not wish to launch any application but perform either a system-wide profile or profile specific set of cores. The screen will look like this:
Profiling a Process

Process(es): Select this if you want to profile an application/process which is already running. This will bring up a process table which can be refreshed. Selecting any one of the process from the table is mandatory in order to start profile.
Selecting Profile Type

Once profile target is selected and configured, clicking Next button will take you to the Select Profile Type fragment. Note that specifying any invalid option will disable the Next button. This fragment lets you to decide what kind of profile data is collected. For example, if you want to see the overall assessment of an application’s runtime behavior, you can select Assess Performance. This lists the predefined profile configurations which contain the relevant PMC events. The screen will look like this:

Custom Profile

If you want to pick and choose the interesting PMC events to be monitored, you can click on Custom Profile. Once clicked, you will be taken to the following fragment:
Selecting any event from the **Select the relevant events to profile** tree will show its corresponding description. You can select one or multiple such events (Ctrl + Click) and then click on the **Add Event** button to add it to the **Monitored Events** table on the right. Removing events is achieved by selecting one or more from the **Monitored Events** table and clicking the **Remove Event** button.

Clicking any row on the **Monitored Events** table will show the corresponding mask for that event with its description. Furthermore, for each event, you can optionally set the mode (User and Kernel) and the sampling interval for the event. Changing the mask is accomplished by checking or un-checking the corresponding mask entries and the table will reflect the value.

In case you want to search for a specific event, just type the name or event-id in the **Filters** text box and a corresponding drop-down will show the matching counter names/event-id.

Note: You can always double-click an event in the **Select the relevant events to profile** tree to add it and double-click on the **Monitored Events** table to remove it. Also, same event with the same mask and mode cannot be added multiple times.

### Setting Advanced Options

Clicking **Advanced Options** button in the bottom of the **Select Profile Type** fragment, will navigate to the **Advanced Options** fragment of **Start Profiling** section. This contains collapsible panes. Related options are grouped together in each of this panes and these options are not compulsory however.
Enable Thread Concurrency Option If you want to have the thread concurrency graph, then select the “Enable Thread Concurrency” option.

Note: This is Windows only option requiring admin privilege.

Call Stack Options Use this to collect call-stack data. By default it is disabled, in order to enable it, click on the option Enable CSS. Once enabled, you can set the “Call Stack Collection” mode, “Call Stack Depth” and “Enable FPO”. Enabling FPO leads to better call-stack collection.

Note:

1. “Call Stack Depth” is not available on Linux.
2. “Call Stack Options” pane will not be displayed for “System” wide profiling.

Symbols and Sources Use “Sources Directory” option to set the path to the source directories (if the sources are relocated) and “Symbols Path” to set the path to the symbol directories where the debug PDB files are available.

Profile Scheduling Use this pane to set scheduling options like the “Profile Duration” and “Start Profiling After”. In case, if Instrumentation APIs are used for fine-grained profiling, the “Are you using Profile Instrumentation API?” option should be enabled.
Start Profile

Once you are done setting the options and when done correctly, the Start Profile green-colored button at the bottom will be enabled and you can click on it to start the profile. After profile initialization, you will see the running timer displaying the number of seconds passed starting from zero. It will look like this:

Here you can either cancel the profile which will take you back to Select Profile Target fragment of HOME page, or stop the profile by clicking “Stop Profiling” button. Once the profiling is stopped, the collected profile data will be processed.
Note: The data translation can take a long time based on options provided during profiling. Enabling call stack collection significantly increases the translation time and so does enabling remote symbol servers.

### 3.4.2 CPU Profile Data Analysis

Once the translation of the raw profile data completes, the profile data can be analyzed through various sections to identify the potential performance bottlenecks.

#### Profile Summary

Once the translation completes, the **SUMMARY** page sections will be populated with the profile data and **Hot Spots** section will be presented. This **SUMMARY** page gives an overview of the hot spots for the profile session through various sections like “Hot Spots” and “Thread Concurrency graph”.

Apart from this, you can also see certain ancillary information regarding profile “Session Information” and “System Information” in which the profiling was carried out.

#### Hot Spots

In this section, hot spots will be shown for functions, modules, process and threads. Process and Threads will only be shown if there are at least two of them. The hot-spots are shown per counter and it can be selected from drop-down in top right corner. Changing it to any other counter will update the hot-spot data accordingly.
The hot-spot donut chart for functions is interactive in nature i.e. you can click on any section and the corresponding function’s source will open in a separate tab in SOURCES page.

**Thread Concurrency graph**

Thread concurrency graph shows the number of threads of a process, running concurrently for the time elapsed (in seconds).
Analyze profile data

Process & Function level Analysis

Click on the ANALYZE button on the top navigation bar to go to counter-wise sample data attributed at various granularities - Process, Load Modules, Threads and Functions. This section will look like this:

The section contains data in two different formats:

1. The upper tree represents counter samples grouped by “Process”. The tree can be expanded to see the child entries for each parent (i.e. for a process). The “Load Modules” and “Threads” are child entries for the selected process entry.

2. The lower table contains function-level counter samples. This data depends on what is selected in the upper tree. For more specific data, you can select a child entry from the upper tree and the corresponding function data will be updated in the lower tree.

Not all entries will be loaded for a profile. To load more than the default entries, click the “Load more functions” or “Load more profile data” buttons on the top right corners to fetch more data. The columns can be sorted as well by clicking on the column headers.

The data can be filtered by expanding the Filters and Options collapsible pane.

1. The View controls the counters that are displayed. The relevant counters and their derived metrics are grouped in predefined views. The user can select the views from the View drop-down.

2. The counter values can be displayed either as absolute count or percentage.

3. Profile data attributed to system modules can be excluded.

4. The Group By drop-down is used to group the data by Process, Module and Thread. By default, the sample data is grouped-by “Process”
Source & Disassembly level Analysis

Double-clicking any entry on the Functions tree will make the GUI load the source tab for that function. If the GUI can find the path to the source for that function, then it will try to open the file failing which you will be prompted to locate it.

Note: The name of the file will be present in the title bar of the browse dialog. If locating and opening the file is successful, it will be displayed in a separate tab in SOURCES page with the file name displayed.
Each row in the source tab can be expanded to see the assembly instructions generated for the corresponding source line. The tree will also show the offset for each assembly instruction along with counter samples.

For multi-threaded/processed applications, if a function has been executed from multiple threads/processes, then each of them will be listed in the processes and threads drop-downs in filters. Changing them will update the counter sample data for that particular selection. By default, data for all process and all thread is shown. Note that selecting a different process from the current selection will reset the “Threads” drop-down to “All Threads” again as each process can spawn multiple threads.

As with the case for counter samples section, there are filters present here as well. You can modify the process, threads, value show type (absolute vs. percentage). This helps you to get more granular data. The function drop-down on the top right helps you to see the source-level attribution for all the functions present in the file which have some samples. Selecting any one of them will take you there. However, the data for the previous one will be removed.

If the source file cannot be located or opened, only disassembly will be displayed.

Call Graph

If the profile was created with Call-stack sampling enabled, the call graph will be displayed in the ANALYZE page and can be navigated by clicking Call Graph Samples.
The data can be browsed based on processes and counters (which can be selected from the drop-down at the top). The top central table displays call-stack samples for each function. Clicking on any function updates the bottom two tables. These tables display the callers and callees respectively of the selected function.

**Instruction Mix Analysis**

If the IBS Fetch and IBS Op are used as sampled events, the instruction mix analysis can be performed by navigating to **IMIX** section in **ANALYZE** page. In this analysis, the number of samples collected for different instruction types.
3.4.3 Importing Profile Databases

Profile databases generated through CLI can be imported in the GUI. For this, from HOME page you can navigate to Open Profile and you will see the following:

While importing, you can specify the DB file and along with that the sources directory for the application which was profiled and symbol servers if applicable. The database format is .db, the raw data format is .caperf on Linux and .pdata or .prd on Windows. These files are recognized by the GUI and can be opened. For raw data, importing will trigger a translation to .db format before opening and thus may take some time. However, if the GUI finds an
already existing translation, it will instead ask your permission to either open the existing translation, re-translate and open or simply cancel it.

Once the translation completes, the database will be opened and you will see the hot-spots.

### 3.4.4 Settings

There are certain application-wide settings to customize the experience. The “SETTINGS” page is located in top-right corner and is divided into three sections, each having a short description of what it contains. The settings once changed can be applied by clicking the “Apply” button. There are settings which are common with profile data filters and hence any change in them when applied through “Apply” button will only get applied to such views which **do not** have local filters set. In case you want to override them, you can click on the “Apply & Override Local Filters” button. (Note: You will lose all local filters applied). You can always reset the settings by clicking “Reset” button or “Cancel” to cancel any changes that you don’t want to apply.
3.4.5 Miscellaneous

Once you have a profile session or imported a profile database, a history is created and the last 50 opened profile databases’ records are stored (i.e. where they are located). Such a list will come up in the Welcome section in the HOME page.

Any profile entry can be opened by clicking it.
Similarly in case of CPU profiles generated (when you set the options and start profiling), if it generates at least
one valid database, the profile will be stored with the options set and can be loaded again in future. Such a list is available in **PROFILE** page in **Saved Configurations** section. Note that by default the profile name is generated by the application and if you want to reuse it, you should ideally name it so that it is easy to locate. This can be done by typing a name in the bottom left corner when setting the profile options.

When you click the “See Details” button to pop-up details.

### 3.5 Limitations

**CPU Profiler Limitations**

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

- In case of Zeppelin B1 parts, only one PMC register is used at a time for the PMC events based profiling.
4.1 Introduction

AMD uProf’s Power Profiler is a tool to analyze the energy efficiency of systems based on AMD CPUs, APUs and dGPUs. It is used to collect and analyze the power, frequency and thermal characteristics data.

4.1.1 Features

- **Reports the following data:**
  - Estimated average energy consumption of Socket and CPU Cores.
  - Estimated average power consumption of supported APUs and dGPUs.
  - Core Effective Frequency of the CPU cores.
  - Average Frequency of the supported iGPUs and dGPUs.
  - Thermal characteristics of the CPU compute-units, iGPUs and dGPUs.
  - CPU Core P-States.
  - Application Analysis

- **Supported Languages for Application Analysis:**
  - C, C++ and Fortran applications

- **Supported Processors:**
  - AMD CPUs: Ryzen, Threadripper, EPYC
  - AMD APUs: Carrizo, Kaveri, Mullins, Temash, Stoney, Bristol
  - AMD dGPUs: Graphics IP 7 GPUs, Radeon and FirePro models

- **Supported Counters:**
  - *Power Counters*

4.2 Key Concepts

This section explains various key concepts related to Power Profiling. The AMD uProf offers live power profile and application power analysis through the Graphical user interface tool **AMDuProf** and Command line user interface tool **AMDuProfCLI**.
4.2.1 Profiling Concepts

Live Power Profile: The Power Profiler follows a statistical sampling-based approach to gather the profile data periodically. The profile data is collected from various hardware counters, PMC (Performance Monitor Counters), SMU (System Management Unit) and PCIe (Peripheral Component Interconnect Express) address space registers. These profile data is plotted in a running timechart graph. This helps users to monitor the current state of the system hardware components. For example, on a Ryzen system, Socket Power, PPT Limit, Energy of each core can be selected together for observation. At every sampling period, profiler will collect the profile data and plot them in a live timechart graph.

Time Chart: Similar to Live Power Profile, CLI provides a way to collect and report the statistically sampled profile data into a text or CSV file format. This option has much lower overhead than the Live Power Profile.

Application Analysis: This analysis can be used to identify the most power intensive application/thread/module/function/source. The profiler attributes the energy consumption to the various applications running in the system, using IPC. This analysis can be used to optimize the application to reduce its energy consumption.

Note: This feature is available only on Windows OS.

4.2.2 Hardware Resources

Running Average Power Limit (RAPL) MSRs: RAPL MSRs provide socket and physical core level energy consumption data. These special registers are available on AMD family 17 processors - Ryzen, Threadripper and EPYC.

System Management Unit (SMU): SMU within the AMD Processor provides interface to retrieve the various power, frequency and thermal characteristics like average CPU Core power, Package power, iGPU power, dGPU power, temperature, frequency etc.

Model Specific Registers: A number of Model Specific Registers are exposed by AMD processor. Details can be found in BDKG/PPR document available in the AMD Developer Web page. The Power Profiler make use of number of MSRs to calculate Core Effective Frequency, P-State and other profile data.

4.3 Command Line Interface (CLI)

AMD uProf’s Power Profiler provides a command line interface utility AMDuProfCLI to collect the power, frequency and thermal characteristics data. This utility is targeted for the users who prefer to use command interpreters like cmd.exe on Windows and bash on Linux. This utility can be used from a batch file or script.

4.3.1 Options

Usage:

```
> AMDuProfCLI.exe [--version] [--help] <command> <options> [application] [arguments]
```

`--version`: Displays the version of AMD uProf tool.
`-h | --help`: Displays the quick help details.

`<application>`: Path to the launch application to be profiled.
`<arguments>`: Arguments (if any) to be passed to the launch application.
**Table 4.1: Following commands are supported:**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>collect</code></td>
<td>Run the given input application and collect power profile samples. Supported only on Windows.</td>
</tr>
<tr>
<td><code>timechart</code></td>
<td>Run power profiler and collect counters for supported combination of device and category types.</td>
</tr>
<tr>
<td><code>report</code></td>
<td>Process the given power profile data file and generate a report in CSV format.</td>
</tr>
<tr>
<td><code>help</code></td>
<td>Opens this HTML help page in a web browser.</td>
</tr>
</tbody>
</table>

**Table 4.2: Following options are supported with ‘collect’ command.**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`-h</td>
<td>--help`</td>
</tr>
<tr>
<td><code>--config &lt;power&gt;</code></td>
<td>Predefined data-collection configuration to be used to collect power samples.</td>
</tr>
<tr>
<td>`-p</td>
<td>--pid &lt;PID,PID,...&gt;`</td>
</tr>
<tr>
<td>`-a</td>
<td>--system-wide`</td>
</tr>
<tr>
<td>`-c</td>
<td>--cpu &lt;core-id,...&gt;`</td>
</tr>
<tr>
<td>`-d</td>
<td>--duration &lt;n&gt;`</td>
</tr>
<tr>
<td><code>--affinity</code></td>
<td>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.</td>
</tr>
<tr>
<td>`-b</td>
<td>--terminate`</td>
</tr>
<tr>
<td>`-s</td>
<td>--start-delay &lt;n&gt;`</td>
</tr>
<tr>
<td>`-w</td>
<td>--working-dir &lt;dir&gt;`</td>
</tr>
<tr>
<td>`-o</td>
<td>--output &lt;file name&gt;`</td>
</tr>
<tr>
<td>`-v</td>
<td>--verbose &lt;n&gt;`</td>
</tr>
</tbody>
</table>

**Table 4.3: Following options are supported with ‘timechart’ command.**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--list</code></td>
<td>Display all the supported devices and categories.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 4.3 – continued from previous page

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| `-e` `-event <type,...>` | Collect counters for specified type or comma separated list of types, where type can be a device or a category.  
**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 Dynamic Voltage and Frequency Scaling (DVFS) counters.  
energy: Collect all available energy counters.  
correlatedpower: Collect all available correlated power counters.  
cac: Collect all available cac counters.  
controllers: Collect all available controllers counters.  
Multiple occurrences of `-e` is allowed. |
| `--histogram` | Collect histogram counters. Allowed only with an occurrence of `-e frequency`. |
| `--cumulative` | Collect cumulative counters. Allowed only with an occurrence of `-e power`. |
| `-t` `-interval <n>` | Sampling interval n in milliseconds. The minimum value is 10ms. |
| `-d` `-duration <n>` | Profile duration n in seconds. |
| `--affinity <core-id,..>` | Core affinity. Comma separated list of core-ids. Ranges of core-ids also be specified, e.g. 0-3. Default affinity is all the available cores. Affinity is set for the launched application. |
| `-w` `-working-dir <dir>` | Set the working directory for the launched target application. |
| `-f` `-format <fmt>` | Output file format. Supported formats are:  
txt: Text (.txt) format.  
csv: Comma Separated Value (.csv) format.  
Default file format is CSV. |
| `-o` `-output <filepath>` | Output file path. |
| `-h` `-help` | Displays this help information. |

Table 4.4: Following options are supported with ‘report’ command.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-h</code> <code>-help</code></td>
<td>Displays this help information.</td>
</tr>
<tr>
<td><code>-r</code> <code>-input &lt;file name&gt;</code></td>
<td>Input file name. Either the raw profile data file (.pdata on Windows) or the processed data file (.db) can be specified.</td>
</tr>
<tr>
<td><code>-o</code> <code>-output &lt;output dir&gt;</code></td>
<td>Output directory in which the processed data file .db, .csv will be created. The default path will be <code>%Temp%\&lt;base-name-of-input-file&gt;.*</code> on Windows and <code>/tmp/&lt;base-name-of-input-file&gt;.*</code> on Linux.</td>
</tr>
<tr>
<td><code>--summary</code></td>
<td>Report only the overview of the profile.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--group-by &lt;section&gt;</td>
<td>Specify the report to be generated. Supported report options are:</td>
</tr>
<tr>
<td></td>
<td>process: Report process details.</td>
</tr>
<tr>
<td></td>
<td>module: Report module details.</td>
</tr>
<tr>
<td></td>
<td>thread: Report thread.</td>
</tr>
<tr>
<td>--src</td>
<td>Generate detailed function report with source statements.</td>
</tr>
<tr>
<td>--disasm</td>
<td>Generate detailed function report with assembly instructions.</td>
</tr>
<tr>
<td>--sort-by &lt;event-index&gt;</td>
<td>Specify the event index on which the reported profile data will be sorted.</td>
</tr>
<tr>
<td>--ignore-system-module</td>
<td>Ignore samples from System Modules.</td>
</tr>
<tr>
<td>--show-percentage</td>
<td>Show Percentage.</td>
</tr>
<tr>
<td>--src-path &lt;path1;..&gt;</td>
<td>Source file directories. (Semicolon separated paths.)</td>
</tr>
<tr>
<td>--symbol-path &lt;path1;..&gt;</td>
<td>Debug Symbol paths. (Semicolon separated paths.)</td>
</tr>
<tr>
<td>--symbol-server &lt;path1;..&gt;</td>
<td>Symbol Server directories. (Semicolon separated paths.)</td>
</tr>
<tr>
<td>--symbol-cache-dir &lt;path&gt;</td>
<td>Path to store the symbols downloaded from the Symbol Servers.</td>
</tr>
<tr>
<td>--cutoff &lt;n&gt;</td>
<td>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.</td>
</tr>
<tr>
<td>-v</td>
<td>--verbose &lt;n&gt;</td>
</tr>
<tr>
<td></td>
<td>1: INFO</td>
</tr>
<tr>
<td></td>
<td>2: DEBUG</td>
</tr>
<tr>
<td></td>
<td>3: EXTENSIVE</td>
</tr>
</tbody>
</table>

### 4.3.2 Examples

#### Timechart Command

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

```bash
> AMDuProfCLI.exe timechart --event power --interval 100 --duration 10
```

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

```bash
> AMDuProfCLI.exe timechart --event frequency -o %Temp%\PowerOutput --interval 500 --duration 10
```

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:

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

#### Collect Command

Launch classic.exe and collect ‘Power’ samples in SWP mode:

```bash
> AMDuProfCLI.exe collect --config power -a -o c:\Temp\pwrprof-swp classic.exe
```

Launch classic.exe and collect ‘Power’ samples for application:
Report Command

Generate report from the raw datafile:

```shell
> AMDuProfCLI.exe report -i c:\Temp\pwrprof.pdata -o c:\Temp\pwrprof-out
```

Generate report with Symbol Server paths:

```shell
> AMDuProfCLI.exe report --symbol-path C:\AppSymbols;C:\DriverSymbols --symbol-server http://msdl.microsoft.com/download/symbols --cache-dir C:\symbols -i c:\Temp\pwrprof.pdata -o c:\Temp\pwrprof-out
```

4.3.3 Sample Report

Following command collects all the correlatedpower counter values for the duration of 1 seconds with sampling interval of 100 milliseconds:

```shell
> AMDuProfCLI.exe timechart --event correlatedpower --interval 100 --duration 1 -o %Temp%\PowerOutput
```

It will generate a timechart report (similar to the below screen-shot) in the file: %Temp%\PowerOutput.csv

![Sample Report](image)

4.4 Graphical User Interface (GUI)

AMDuProf’s Power Profiler provides following functionalities:

- “Live Power Profile” to collect and report the power, frequency and thermal data in live graphs.
- “Power App Analysis” for application Analysis to identify the energy intensive applications, load modules, functions and source lines.
4.4.1 Live Power Profile

AMDuProf GUI lets the user to select the counters based on the available counter categories, sampling interval, launched application to be profiled and other related configurations. Based on the user input, GUI plots the collected profile data and display the data in live graphs.

Starting a Live Power Profile

There are two ways to start a new power profile session from the HOME page below.

- For quick profiling click See what’s guzzling power in your system link. This will select the System-wide Power Profile (Live) option in Select Profile Type fragment of “Start Profiling” section.
- An advanced user can click Create a new profile? link under the heading “Start Here” and proceed to “Select Profile Target” and “Select Profile Type” fragments for further configurations.

Selecting Profile Target

You can see Select Profile Target fragment in the Start Profiling section. In this fragment, different types of profile target can be selected from the drop-down list. Following profile target options are available:

Profiling an Application

Application: Select this target when you want to launch an application and profile it (or launch and do a system-wide profile). The only compulsory option is a valid path to the executable. (By default, the path to the executable becomes the working directory unless you specify a path). Once provided, it will look like this:
**Profiling the System**

**System**: Select this if you do not wish to launch any application but perform either a system-wide profile or profile specific set of cores. The screen will look like this:
Selecting Live Counters

Once profile target is selected and configured, clicking Next button will take you to the Select Profile Type fragment. Note that specifying any invalid option will disable the Next button. This fragment lets you to decide what kind of profile data to be collected. For live power profile, select System-wide Power Profile (Live) option in the drop-down list.

Counters are arranged based on categories such as energy, power, frequency, temperature etc. Within a category, counters are arranged in a tree structure based on the device hierarchy. User can select or unselect any number of counters by checking or unchecking the corresponding check-box adjacent to the counter name.

There should be at least one counter selected to start the profiling.

Time Chart

Once the interesting counters are selected, clicking Start Profile button will start the data collection and plot the data in graphs. There is also a corresponding data table adjacent to each graph to display the current value of the counters.
Line graphs are grouped together and plotted based on the category. However, if too many counters are selected from the same category, counters are grouped together based on category and the devices they belong to.

**When plotting is in progress, user can:**

- Pause the graphs without pausing the data collection by clicking **Pause Graphs** button, later graphs can be resumed by clicking **Play Graphs** button.
- Stop the profiling without closing the view by clicking the **Stop Profiling** button. This will stop collecting the profile data.
- Stop the profiling and close the view by clicking **Close View** button

When plotting is in stopped state, to analyze the plotted data user can hover the mouse over the graph to view the corresponding data values of the counters in the data table.

### 4.4.2 Application Analysis

The **Power App Analysis** option in **Select Profile Type** section is used to perform application analysis of energy consumption of a launched application or all the running applications in the system. It can be used to identify the energy intensive functions with respect to the energy consumption of the application or for all applications running in system during the profile run. User can drill down to source code level, provided required symbol information and source files are available for the target application. Further, user can also attach a specific process running from a list of running processes.

Power profile application analysis and CPU profile application analysis follow the common GUI work flow. Please refer to the “Graphical User Interface” section of “CPU Profiler” in this document.

### 4.4.3 Settings

**Save Live Data** Live data can be saved in a SQLite db file for future use. This option will enable writing the profile data into a database file. This database file can be imported in GUI to analyze the profile data.

**Save Live Data at** User can choose a path to store the output database file.
**Sampling Interval** Sampling interval implies the interval at which profiler will collect data from driver and plot them in the graph. Default value is 100ms. It is recommended to choose a larger sampling period to avoid overhead of data collection from the driver. Please note profiler may not allow the system to transition to sleep state if the sampling interval is too small.

4.5 Limitations

**Power Profiler Limitations**

- Only one profile session can be run at a time.
- Please make sure latest Radeon driver is installed before running power profiler. Newer version of dGPU may go to sleep (low power) state frequently if there is no activity in dGPU. In that case, power profiler may emit a warning AMDT_WARN_SMU_DISABLED. Counters may not be accessible in this state. Before running the power profiler, it is advisable to bring the dGPU to active state.
- ICELAND dGPU(Topaz-XT, Topaz PRO, Topaz XTL, Topaz LE) series is not supported.
- Application Analysis is supported only on Windows OS.
- Minimum supported sampling period in CLI is 10ms. Whereas in GUI, for the platforms other than EPYC, the minimum supported sampling period is 100ms and for EPYC it is 500ms. It is recommended to use large sampling period to reduce the sampling and rendering overhead.
- If SMU is not accessible while profiling is in progress, behavior may be undefined.
- Power profile application analysis is supported only on the Windows platforms.
5.1 Introduction

Performance Counter Monitoring utility AMDuProfPcm provides basic performance monitoring metrics for AMD’s family 17h processors. This utility periodically collects the CPU Core, L3, DF and UMC’s PMC events count values and report metrics on instructions per cycle, core effective frequency, cache misses and memory bandwidth etc.

This utility is support only on Linux.

5.1.1 Features

Following default metrics are collected and reported by this utility:

- Core Utilization
- Core Effective Frequency (w/o halt)
- Core Effective Frequency (w/ halt)
- IPC (Sys + User)
- CPI (Sys + User)
- L2 Request
- L2 DC Request
- L2 DC Miss
- L3 Access
- L3 Hit
- L3 Miss
- L3 CCX Miss

5.2 Command Line Interface (CLI)

Usage: AMDuProfPcm [<options>]

5.2.1 Options

Following are the list of supported options.
Table 5.1: Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c &lt;core</td>
<td>ccx</td>
</tr>
<tr>
<td>-m &lt;cpu</td>
<td>cache</td>
</tr>
<tr>
<td>-d &lt;seconds&gt;</td>
<td>Profile duration to run.</td>
</tr>
<tr>
<td>-t &lt;interval in ms&gt;</td>
<td>Interval in which pmc count values will be read. Default is 1000ms. Minimum is 1000ms.</td>
</tr>
<tr>
<td>-o &lt;output file&gt;</td>
<td>Output file name.</td>
</tr>
<tr>
<td>-p &lt;n&gt;</td>
<td>Set precision of the metrics reported. Default is 2.</td>
</tr>
<tr>
<td>-r</td>
<td>To force reset the MSRs.</td>
</tr>
<tr>
<td>-i &lt;config file&gt;</td>
<td>XML config file that specifies CoreL3 counters to monitor.</td>
</tr>
<tr>
<td>-v</td>
<td>Print version.</td>
</tr>
<tr>
<td>-h</td>
<td>Print help.</td>
</tr>
</tbody>
</table>

5.2.2 Examples

Monitor all the supported counters on all the Cores/CCX/Nodes for the duration of 10 seconds and save the profile data:

```
# ./AMDuProfPcm -d 10 -o /tmp/pcm-overview.csv
```

Monitor core 0 for the duration of 10 seconds and save the profile data in a file /tmp/perf-overview.csv:

```
# ./AMDuProfPcm -c core=0 -d 10 -o /tmp/pcm-overview.csv
```

Monitor all the supported metrics for 60 seconds and save the output in /tmp/pcm.csv. This will collect Core PMC metrics:

```
# ./AMDuProfPcm -d 60 -c core=0 -o /tmp/pcm.csv
```

Monitor all the supported metrics for the given node and its ccx and cores:

```
# ./AMDuProfPcm -d 60 -c die=0 -o /tmp/pcm.csv
```

Monitor all the supported metrics for the given node and its ccx and cores:

```
# ./AMDuProfPcm -d 60 -c die=0 -o /tmp/pcm.csv
```

Monitor only the memory bandwidth across all the UMCs for the duration of 60 seconds

```
# ./AMDuProfPcm -d 60 -m memory -o /tmp/pcm.csv
```

Monitor only the memory bandwidth for a particular die for the duration of 60 seconds

```
# ./AMDuProfPcm -d 60 -c die=0 -m memory -o /tmp/pcm.csv
```

Print help

```
# ./AMDuProfPcm -h
```
AMD uProf provides following development libraries for the developers:

### 6.1 AMDPowerProfileAPI Library

#### 6.1.1 Introduction

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

#### 6.1.2 APIs

**AMDPwrProfileInitialize**

This API loads and initializes the AMDT Power Profile drivers. This API should be the first one to be called.

```c
AMDTResult AMDPwrProfileInitialize (AMDPwrProfileMode profileMode)
```

**Parameters**

- `profileMode`: Client should select any one of the predefined profile modes that are defined in `AMDPwrProfileMode`.

**Returns**

- `AMDT_STATUS_OK`: Success
- `AMDT_ERROR_INVALIDARG`: An invalid `profileMode` parameter was passed
- `AMDT_ERROR_DRIVER_UNAVAILABLE`: Driver not available
- `AMDT_DRIVER_VERSION_MISMATCH`: Mismatch between the expected and installed driver versions
- `AMDT_ERROR_PLATFORM_NOT_SUPPORTED`: Platform not supported
- `AMDT_WARN_SMU_DISABLED`: SMU is disabled and hence power and thermal values provided by SMU will not be available
- `AMDT_WARN_IGPU_DISABLED`: Internal GPU is disabled
- `AMDT_ERROR_FAIL`: An internal error occurred
• AMDT_ERROR_PREVIOUS_SESSION_NOT_CLOSED: Previous session was not closed.

AMDTpwrGetSupportedCounters

This API provides the list of counters supported by the platform. The pointers returned will be valid till the client calls AMDTpwrProfileClose() function.

```c
AMDTResult AMDTpwrGetSupportedCounters(AMDTUInt32 *pNumCounters, AMDTPwrCounterDesc **ppCounterDescs)
```

**Parameters**

• pNumCounters: Number of counters supported by the device
• ppCounterDescs: Description of each counter supported by the device

**Returns**

• AMDT_STATUS_OK: On Success
• AMDT_ERROR_INVALIDARG: NULL pointer was passed as ppCounterDescs or pNumCounters parameters
• AMDT_ERROR_DRIVER_UNINITIALIZED: AMDTpwrProfileInitialize() function was neither called nor successful
• AMDT_ERROR_INVALID_DEVICEID: invalid deviceId parameter was passed
• AMDT_ERROR_OUTOFMEMORY: Failed to allocate required memory
• AMDT_ERROR_FAIL: An internal error occurred

AMDTpwrGetCounterId

This API provides the counter id for a basic counter.

```c
AMDTResult AMDTpwrGetCounterId(AMDTCounter counter, AMDTUInt32 *pCounterId)
```

**Parameters**

• counter: supported counter to get the counter id
• pCounterId: counter id of counter.

**Returns**

• AMDT_STATUS_OK: On Success
• AMDT_ERROR_INVALIDARG: NULL pointer was passed as pCounterDesc parameter
• AMDT_ERROR_DRIVER_UNINITIALIZED: AMDTpwrProfileInitialize() function was neither called nor successful
• AMDT_ERROR_INVALID_COUNTERID: Invalid counterId parameter was passed
• AMDT_ERROR_FAIL: An internal error occurred
**AMDPwrGetCounterDesc**

This API provides the description for the given counter index.

```plaintext
AMDTResult AMDPwrGetCounterDesc(AMDTUInt32 counterId, AMDPwrCounterDesc *pCounterDesc)
```

**Parameters**

- **counterId**: Counter index
- **pCounterDesc**: Description of the counter which index is counterId

**Returns**

- **AMDT_STATUS_OK**: On Success
- **AMDT_ERROR_INVALIDARG**: NULL pointer was passed as pCounterDesc parameter
- **AMDT_ERROR_DRIVER_UNINITIALIZED**: AMDTPwrProfileInitialize() function was neither called nor successful
- **AMDT_ERROR_INVALID_COUNTERID**: Invalid counterId parameter was passed
- **AMDT_ERROR_FAIL**: An internal error occurred

**AMDPwrEnableCounter**

This API will enable the counter to be sampled. This API cannot be used once profile is started.

```plaintext
AMDTResult AMDPwrEnableCounter(AMDTUInt32 counterId)
```

**Parameters**

- **counterId**: Counter index

**Returns**

- **AMDT_STATUS_OK**: On Success
- **AMDT_ERROR_DRIVER_UNINITIALIZED**: AMDTPwrProfileInitialize() function was neither called nor successful
- **AMDT_ERROR_INVALID_COUNTERID**: Invalid counterId parameter was passed
- **AMDT_ERROR_COUNTER_ALREADY_ENABLED**: Specified counter is already enabled
- **AMDT_ERROR_PROFILE_ALREADY_STARTED**: Counters cannot be enabled on the fly when the profile is already started
- **AMDT_ERROR_PREVIOUS_SESSION_NOT_CLOSED**: Previous session was not closed
- **AMDT_ERROR_COUNTER_NOT_ACCESSIBLE**: Counter is not accessible
- **AMDT_ERROR_FAIL**: An internal error occurred

**AMDPwrSetTimerSamplingPeriod**

This API will set the driver to periodically sample the counter values and store them in a buffer. This cannot be called once the profile run is started. This API is not required to call if AMDTPwrProfileInitialize API is called with AMDT_PWR_MODE_INSTANT_COUNTER as profileMode.
AMDTResult AMDTPwrSetTimerSamplingPeriod(AMDTUInt32 interval)

Parameters

- interval: sampling period in millisecond

Returns

- AMDT_STATUS_OK: On Success
- AMDT_ERROR_INVALIDARG: Invalid interval value was passed
- AMDT_ERROR_DRIVER_UNINITIALIZED: AMDTPwrProfileInitialize() function was neither called nor successful
- AMDT_ERROR_PROFILE_ALREADY_STARTED: Timer interval cannot be changed when the profile is already started
- AMDT_ERROR_PREVIOUS_SESSION_NOT_CLOSED: Previous session was not closed
- AMDT_ERROR_FAIL: An internal error occurred

AMDTResult AMDTPwrStartProfiling()

This API will start the profiling and the driver will collect the data at regular interval specified by AMDTPwrSetTimerSamplingPeriod(). This has to be called after enabling the required counters by using AMDTPwrEnableCounter().

Returns

- AMDT_STATUS_OK: On Success
- AMDT_ERROR_DRIVER_UNINITIALIZED: AMDTPwrProfileInitialize() function was neither called nor successful
- AMDT_ERROR_TIMER_NOT_SET: Sampling timer was not set
- AMDT_ERROR_COUNTERS_NOT_ENABLED: No counter enabled for collecting profile data
- AMDT_ERROR_PROFILE_ALREADY_STARTED: Profile is already started
- AMDT_ERROR_PREVIOUS_SESSION_NOT_CLOSED: Previous session was not closed
- AMDT_ERROR_BIOS_VERSION_NOT_SUPPORTED: BIOS needs to be upgraded
- AMDT_ERROR_FAIL: An internal error occurred
- AMDT_ERROR_ACCESSDENIED: Profiler is busy, currently not accessible

AMDTResult AMDTPwrReadAllEnabledCounters(AMDTUInt32 *pNumOfSamples, AMDTPwrSample **pData)

This API will read all the counters that are enabled. This can return an array of {CounterID, Float-Value}. If there are no new samples, this API will return AMDT_ERROR_PROFILE_DATA_NOT_AVAILABLE and pNumOfSamples will point to value of zero. If there are new samples, this API will return AMDT_STATUS_OK and pNumOfSamples will point to value greater than zero.
Parameters

- **pNumOfSamples**: Number of sample based on the AMDTPwrSetSampleValueOption() set
- **ppData**: Processed profile data. No need to allocate or free the memory data is valid till we call this API next time

Returns

- **AMDT_STATUS_OK**: On Success
- **AMDT_ERROR_INVALIDARG**: NULL pointer was passed as pNumSamples of ppData parameters
- **AMDT_ERROR_DRIVER_UNINITIALIZED**: `AMDTPwrProfileInitialize()` function was neither called nor successful
- **AMDT_ERROR_PROFILE_NOT_STARTED**: Profile is not started
- **AMDT_ERROR_PROFILE_DATA_NOT_AVAILABLE**: Profile data is not yet available
- **AMDT_ERROR_OUTOFMEMORY**: Memory not available
- **AMDT_ERROR_SMU_ACCESS_FAILED**: One of the configured SMU data accessible
- **AMDT_ERROR_FAIL**: An internal error occurred

**AMDTPwrStopProfiling**

This APIs will stop the profiling run which was started by `AMDTPwrStartProfiling()` function call.

```
AMDTResult AMDTPwrStopProfiling()
```

Returns

- **AMDT_STATUS_OK**: On Success
- **AMDT_ERROR_DRIVER_UNINITIALIZED**: `AMDTPwrProfileInitialize()` function was neither called nor successful
- **AMDT_ERROR_PROFILE_NOT_STARTED**: Profile is not started
- **AMDT_ERROR_FAIL**: An internal error occurred
- **AMDT_STATUS_OK**: On Success
- **AMDT_ERROR_FAIL**: An internal error occurred
- **AMDT_ERROR_DRIVER_UNINITIALIZED**: `AMDTPwrProfileInitialize()` function was neither called nor successful

**AMDTPwrProfileClose**

This API will close the power profiler and unregister driver and cleanup all memory allocated during `AMDTPwrProfileInitialize()`.

```
AMDTResult AMDTPwrProfileClose()
```

Returns

- **AMDT_STATUS_OK**: On Success
- **AMDT_ERROR_FAIL**: An internal error occurred
• AMDT_ERROR_DRIVER_UNINITIALIZED: `AMDTPwrProfileInitialize()` function was neither called nor successful

6.1.3 Data Types

**AMDPwrProfileMode**

enum `AMDPwrProfileMode`

Following power profile modes are supported.

- `AMDT_PWR_PROFILE_MODE_ONLINE`: Counter values are collected at every specified sampling interval.
- `AMDT_PWR_MODE_INSTANT_COUNTER`: Counter values are collected instantly.

Note: `AMDT_PWR_MODE_INSTANT_COUNTER` mode is supported only for AMD Family 17h Model 10h based processor family

**AMDTCounter**

enum `AMDTCounter`

Following power profile counters are supported for `AMDTPwrGetCounterId()`.

- `AMD_PWR_SOCKET_POWER`: Socket Power
- `AMD_PWR_SOCKET_STAPM_LIMIT`: Socket Stapm Limit
- `AMD_PWR_SOCKET_PPT_FAST_LIMIT`: Fast PPT Limit
- `AMD_PWR_SOCKET_PPT_SLOW_LIMIT`: Slow PPT Limit

**AMDPwrUnit**

enum `AMDPwrUnit`

Following are the various unit types for the output values for the counter types.

- `AMDT_PWR_UNIT_TYPE_COUNT`: Count index
- `AMDT_PWR_UNIT_TYPE_PERCENT`: Percentage
- `AMDT_PWR_UNIT_TYPE_RATIO`: Ratio
- `AMDT_PWR_UNIT_TYPE_MILLI_SECOND`: Time in milli seconds
- `AMDT_PWR_UNIT_TYPE_JOULE`: Energy consumption
- `AMDT_PWR_UNIT_TYPE_WATT`: Power consumption
- `AMDT_PWR_UNIT_TYPE_VOLT`: Voltage
- `AMDT_PWR_UNIT_TYPE_MILLI_AMPERE`: Current
- `AMDT_PWR_UNIT_TYPE_MEGA_HERTZ`: Frequency
- `AMDT_PWR_UNIT_TYPE_CENTIGRADE`: Temperature

**AMDTResult**

type `AMDTResult`

`typedef unsigned int AMDTResult`
**AMDPwrCounterDesc**

```go
type AMDPwrCounterDesc
```

struct *AMDPwrCounterDesc* encapsulate details of a supported power counter and its associated device.

**Data Members**

- `AMDTUInt32 m_counterID`: Counter index
- `AMDTUInt32 m_deviceId`: Device Id
- `AMDTDeviceType m_devType`: Device type- Package/Die/Compute unit/Core/dGPU
- `AMDTUInt32 m_devInstanceId`: Device instance id within the device type
- `char *m_description`: Name of the counter
- `char *m_name`: Description of the counter
- `AMDPwrCategory m_category`: Power/Frequency/Temperature
- `AMDPwrAggregation m_aggregation`: Single/Histogram/Cumulative
- `AMDPwrUnit m_units`: Seconds/MHz/Joules/Watts/Volt/Ampere
- `AMDTUInt32 m_parentCounterId`: If the counter has some child counters

**AMDPwrSample**

```go
type AMDPwrSample
```

struct *AMDPwrSample* encapsulate output sample with timestamp and the counter values for all the enabled counters.

**Data Members**

- `AMDPwrSystemTime m_systemTime`: Start time of Profiling
- `AMDTUInt64 m_elapsedTimeMs`: Elapsed time in milliseconds - relative to the start time of the profile
- `AMDTUInt64 m_recordId`: Record id
- `AMDTUInt32 m_numOfCounter`: Number of counter values available
- `AMDPwrCounterValue m_counterValues`: List of counter values

**AMDPwrSystemTime**

```go
type AMDPwrSystemTime
```

struct *AMDPwrSystemTime* represents the system time in second and milliseconds

**Data Members**

- `AMDTUInt64 m_second`: Seconds
- `AMDTUInt64 m_microSecond`: Milliseconds
**AMDPwrCounterValue**

*type AMDPwrCounterValue*

struct *AMDPwrCounterValue* represents a counter id and its value

### Data Members

```c
AMDTUInt32 m_counterID; // Counter index
AMDTUInt32 m_valueCnt; // Number of value for this counter
union
{
    AMDTFloat32 m_data; // Counter value
    AMDTFloat32 *m_pData; // Pointer to the multi value array
}
```

### 6.1.4 Examples

```c
//=============================================================
// (c) 2017 Advanced Micro Devices, Inc.
//
/// \author AMDuProf Developer Tools
/// \brief Example program using the AMDPowerProfile APIs.
//=============================================================

// - Start the profiling
// - Periodically read the counter values and report till the user has
// requested to stop

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>
#include <string.h>
#ifdef __linux__
    #include <unistd.h>
#endif
#include <AMDPowerProfileApi.h>

void CollectAllCounters()
{
    AMDTResult hResult = AMDT_STATUS_OK;
    // Initialize online mode
    hResult = AMDPwrProfileInitialize(AMDT_PWR_MODE_TIMELINE_ONLINE);
    // --- Handle the error

    // Configure the profile run
    // 1. Get the supported counters
    // 2. Enable all the counter
    // 3. Set the timer configuration

    // 1. Get the supported counter details
    AMDTUInt32 nbrCounters = 0;
    AMDTPwrCounterDesc* pCounters = nullptr;
```
hResult = AMDTPwrGetSupportedCounters(&nbrCounters, &pCounters);
assert(AMDT_STATUS_OK == hResult);

AMDTpwrCounterDesc* pCurrCounter = pCounters;

for (AMDTUInt32 cnt = 0; cnt < nbrCounters; cnt++, pCurrCounter++)
{
    if (nullptr != pCurrCounter)
    {
        // Enable all the counters
        hResult = AMDTPwrEnableCounter(pCurrCounter->m_counterID);
        assert(AMDT_STATUS_OK == hResult);
    }
}

// Set the timer configuration
AMDTUInt32 samplingInterval = 100; // in milliseconds
AMDTUInt32 profilingDuration = 10; // in seconds

hResult = AMDTPwrSetTimerSamplingPeriod(samplingInterval);
assert(AMDT_STATUS_OK == hResult);

// Start the Profile Run
hResult = AMDTPwrStartProfiling();
assert(AMDT_STATUS_OK == hResult);

// Collect and report the counter values periodically
// 1. Take the snapshot of the counter values
// 2. Read the counter values
// 3. Report the counter values

volatile bool isProfiling = true;
bool stopProfiling = false;
AMDTUInt32 nbrSamples = 0;

while (isProfiling)
{
    // sleep for refresh duration - at least equivalent to the
    // sampling interval specified
    #if defined ( WIN32 )
    // Windows
    Sleep(samplingInterval);
    #else
    // Linux
    usleep(samplingInterval * 1000);
    #endif

    // read all the counter values
    AMDTPwrSample* pSampleData = nullptr;

    hResult = AMDTPwrReadAllEnabledCounters(&nbrSamples, &pSampleData);

    if (AMDT_STATUS_OK != hResult)
    {
        continue;
    }

    if (nullptr != pSampleData)
    {
        // iterate over all the samples and report the sampled counter values
        for (AMDTUInt32 idx = 0; idx < nbrSamples; idx++)
        {
            // Iterate over the sampled counter values and print
        }
    }
}
for (unsigned int i = 0; i < pSampleData[idx].m_numOfCounter; i++)
{
    if (nullptr != pSampleData[idx].m_counterValues)
    {
        AMDUInt32 id = 0;
        id = pSampleData[idx].m_counterValues->m_counterID;
        // Get the counter descriptor to print the counter
        // name
        AMDPwrCounterDesc counterDesc;
        AMDPwrGetCounterDesc(id, &counterDesc);

        fprintf(stdout, "%s : %f ",
                counterDesc.m_name,
                pSampleData[idx].m_counterValues->m_data);

        pSampleData[idx].m_counterValues++;
    }
}

// iterate over the sampled counters

// iterate over all the samples collected

// check if we exceeded the profile duration
if ((profilingDuration > 0)
    && (pSampleData->m_elapsedTimeMs >= (profilingDuration * 1000)))
{
    stopProfiling = true;
}

if (stopProfiling)
{
    // stop the profiling
    hResult = AMDPwrStopProfiling();
    assert(AMDT_STATUS_OK == hResult);
    isProfiling = false;
}

// Close the profiler
hResult = AMDPwrProfileClose();
assert(AMDT_STATUS_OK == hResult);
}

int main()
{
    AMDTResult hResult = AMDT_STATUS_OK;
    CollectAllCounters();
    return hResult;
}

6.1.5 How to use API Library

Example code must use AMDTPowerProfileAPI.dll to compile and run the example program. We must need to make sure drivers are up and running.

To build and execute a example application, following steps should be performed on Linux machine.

1. Example CollectAllCounters is located at `<AMDuProf-install-dir>/Examples/CollectAllCounters`

    $ cd <AMDuProf-install-dir>/Examples/CollectAllCounters
2. Set LD_LIBRARY_PATH
   
   $ export LD_LIBRARY_PATH=<AMDuProf-install-dir>/bin

3. Compile application code
   
   $ g++ -O -std=c++11 CollectAllCounters.cpp
   -I<AMDuProf-install-dir>/include
   -l AMDPowerProfileAPI -L<AMDuProf-install-dir>/bin
   -Wl,-rpath <AMDuProf-install-dir>/bin -o CollectAllCounters

4. Execute
   
   $ ./CollectAllCounters

6.2 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.

6.2.1 Required File Paths

All the paths mentioned below are with respect to uProf installation directory (UPROF-INSTALL-DIR).

Header File:

<UPROF-INSTALL-DIR>/include/AMDProfileController.h (on Windows platforms)
<UPROF-INSTALL-DIR>/include/AMDProfileController.h (on Linux platforms)

Static Library Files on Windows platforms:

<UPROF-INSTALL-DIR>/lib/x64/AMDProfileController.lib
<UPROF-INSTALL-DIR>/lib/x86/AMDProfileController.lib

Static Library File on Linux platforms:

<UPROF-INSTALL-DIR>/lib/x64/libAMDProfileController.a

6.2.2 The control APIs

To resume CPU profiling, call the below API.

   bool amdProfileResume(AMD_PROFILE_CPU);

To Pause CPU profiling, call the below two API.

   bool amdProfilePause(AMD_PROFILE_CPU);

CPU Profiler only profiles the code within each Resume-Pause APIs pair. After adding these APIs, compile your target application and profile only the desired part of code.

6.2.3 Calling the APIs

Include the header file AMDProfileController.h, and call the resume and pause APIs within the code. The code encapsulated within resume-pause API pair will be profiled by CPU Profiler. These APIs can be called multiple times to profile different parts of the code. These API calls can be spread across multiple functions, i.e. resume...
called from one function and stop called from another function. These APIs can be spread across threads, i.e. resume called from one thread and stop called from another thread of the same target application.

In the below code snippet, the CPU Profiler is restricted to the execution of multiply_matrices() function.

```c
#include <AMDProfileController.h>

int main(int argc, char* argv[]) {
    // Initialize the matrices
    initialize_matrices();

    // Resume the CPU profiler
    amdProfileResume(AMD_PROFILE_CPU);

    // Multiply the matrices
    multiply_matrices();

    // Stop the CPU Profiler
    amdProfilePause(AMD_PROFILE_CPU);

    return 0;
}
```

### 6.2.4 Compiling the target application

To compile the application on Microsoft Visual Studio, update the configuration properties to include the path of header file, LIB file of AMDProfileController.

To compile a C++ application on Linux using G++, use the following command:

```bash
$ g++ -std=c++11 <source-file.cpp> -I<UPROF-INSTALL-DIR>/include -L<UPROF-INSTALL-DIR>/lib/x64/ -lAMDProfileController -lrt -pthread
```

Note: Do not use `-static` option while compiling with G++.

### 6.2.5 Profiling with target application and control APIs

After the compiling the target application, create a project in uProf using it, set the desired CPU profile session options. While setting the CPU profile session options, in the ‘Profile Scheduling’ section, select ‘Are you using Profile Instrumentation API’.

Once all the settings done, start the CPU profiling. The CPU Profiler will be in pause state and target application execution begins. When the resume API gets called from target application, CPU Profile starts profiling till stop API gets called from target application. As soon as stop API is called in target application, CPU Profiler stops profiling and waits for next control API call.

To profile from CLI, option `--start-paused` should be used to start the profiler in pause state.

Sample command on Windows platforms:

```bash
> AMDuProfCLI.exe collect --config tbp --start-paused -o C:\Temp\cpuprof-tbp <target-application.exe>
```

Sample command on Linux platforms:

```bash
$ ./AMDuProfCLI collect --config tbp --start-paused -o /tmp/cpuprof-tbp <target-application>
```
7.1 CPU Performance Metrics

7.1.1 Core PMC Events

Some of the interesting Core Performance events of AMD Processor family 17h are listed here.
Table 7.1: Core PMC Events

<table>
<thead>
<tr>
<th>PMC Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[PMCx076]</td>
<td>CPU clock cycles not halted</td>
</tr>
<tr>
<td>[PMCx0C0]</td>
<td>Retired Instructions</td>
</tr>
<tr>
<td>[PMCx0C1]</td>
<td>Retired uOps</td>
</tr>
<tr>
<td>[PMCx0C2]</td>
<td>Retired Branch Instructions</td>
</tr>
<tr>
<td>[PMCx0C3]</td>
<td>Retired Branch Instructions Mispredicted</td>
</tr>
<tr>
<td>[PMCx0C4]</td>
<td>Retired Taken Branch Instructions</td>
</tr>
<tr>
<td>[PMCx0CA]</td>
<td>Retired Indirect Branch Instructions Mispredicted</td>
</tr>
<tr>
<td>[PMCx08A]</td>
<td>L1 BTB Correction</td>
</tr>
<tr>
<td>[PMCx08B]</td>
<td>L2 BTB Correction</td>
</tr>
<tr>
<td>[PMCx040]</td>
<td>Data Cache Accesses</td>
</tr>
<tr>
<td>[PMCx041]</td>
<td>MAB Allocation by Pipe</td>
</tr>
<tr>
<td>[PMCx043]</td>
<td>Demand Data Cache Fills by Data Source</td>
</tr>
<tr>
<td>[PMCx045]</td>
<td>L1 DTLB Miss</td>
</tr>
<tr>
<td>[PMCx047]</td>
<td>Misaligned loads</td>
</tr>
<tr>
<td>[PMCx080]</td>
<td>32 Byte Instruction Cache Fetched</td>
</tr>
<tr>
<td>[PMCx081]</td>
<td>32 Byte Instruction Cache Misses</td>
</tr>
<tr>
<td>[PMCx084]</td>
<td>L1 ITLB Miss, L2 ITLB Hit</td>
</tr>
<tr>
<td>[PMCx085]</td>
<td>L1 ITLB Miss, L2 ITLB Miss</td>
</tr>
</tbody>
</table>

7.1.2 CPU Metrics

Core Utilization The percentage of time the core was running - i.e non-idle time.

Core Effective Frequency (w/o halt) Core Effective Frequency (without halted cycles) over the sampling period, reported in GHz. The metric is based on APERF and MPERF MSRs. MPERF is incremented by the core at the P0 state frequency while the core is in C0 state. APERF is incremented in proportion to the actual number of core cycles while the core is in C0 state.

Core Effective Frequency (w/ halt) Core Effective Frequency (with halted cycles) over the sampling period, reported in GHz.

IPC (Sys + User) Instruction Retired Per Cycle (IPC) is the average number of instructions retired per cycle. This is measured using Core PMC events PMCx0C0 [Retired Instructions] and PMCx076 [CPU Clocks not
Halted]. These PMC events are counted in both OS and User mode.

**CPI (Sys + User)** Cycles Per Instruction Retired (CPI) is the multiplicative inverse of IPC metric. This is one of the basic performance metric indicating how cache misses, branch mis-predictions, memory latencies and other bottlenecks are affecting the execution of an application. Lower CPI value is better.

**L2 Request** Number of instructions requested to access L2 cache. This metric is in Per Thousand Instructions (PTI).

**L2 DC Request** Number of instructions requested to access L2 data cache. This metric is in Per Thousand Instructions (PTI)

**L2 DC Miss** Number of loads that misses the L2 data cache. This metric is in Per Thousand Instructions (PTI).

### 7.1.3 L3 cache Metrics

**L3 Access** Number of instructions requested L3 cache access. This metric is in Per Thousand Instructions (PTI)

**L3 Hit** Number of instructions that lead to L3 cache hit. This metric is in Per Thousand Instructions (PTI)

**L3 Miss** Number of instructions that lead to L3 cache miss. This metric is in Per Thousand Instructions (PTI).

This metric includes the “neighbor L2 hit” + “missed the L3 cache in the entire CPU Complex (CCX)”.

**L3 CCX Miss** Number of instructions that lead to L3 cache miss in the entire CPU Complex (CCX). This metric is in Per Thousand Instructions (PTI).

### 7.1.4 IBS Derived Events

uProf translates the IBS information produced by the hardware into derived event sample counts that resemble EBP sample counts. All IBS-derived events have “IBS” in the event name and abbreviation. Although IBS-derived events and sample counts look similar to EBP events and sample counts, the source and sampling basis for the IBS event information are different.

Arithmetic should never be performed between IBS derived event sample counts and EBP event sample counts. It is not meaningful to directly compare the number of samples taken for events that represent the same hardware condition. For example, fewer IBS DC miss samples is not necessarily better than a larger quantity of EBP DC miss samples.
Table 7.2: IBS Fetch derived events list

<table>
<thead>
<tr>
<th>IBS Fetch Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All IBS fetch samples</td>
<td>The number of all IBS fetch samples. This derived event counts the number of all IBS fetch samples that were collected including IBS-killed fetch samples.</td>
</tr>
<tr>
<td>IBS fetch killed</td>
<td>The number of IBS sampled fetches that were killed fetches. A fetch operation is killed if the fetch did not reach ITLB or IC access. The number of killed fetch samples is not generally useful for analysis and are filtered out in other derived IBS fetch events (except Event Select 0xF000 which counts all IBS fetch samples including IBS killed fetch samples.)</td>
</tr>
<tr>
<td>IBS fetch attempted</td>
<td>The number of IBS sampled fetches that were not killed fetch attempts. This derived event measures the number of useful fetch attempts and does not include the number of IBS killed fetch samples. This event should be used to compute ratios such as the ratio of IBS fetch IC misses to attempted fetches. The number of attempted fetches should equal the sum of the number of completed fetches and the number of aborted fetches.</td>
</tr>
<tr>
<td>IBS fetch completed</td>
<td>The number of IBS sampled fetches that completed. A fetch is completed if the attempted fetch delivers instruction data to the instruction decoder. Although the instruction data was delivered, it may still not be used (e.g., the instruction data may have been on the “wrong path” of an incorrectly predicted branch.)</td>
</tr>
<tr>
<td>IBS fetch aborted</td>
<td>The number of IBS sampled fetches that aborted. An attempted fetch is aborted if it did not complete and deliver instruction data to the decoder. An attempted fetch may abort at any point in the process of fetching instruction data. An abort may be due to a branch redirection as the result of a mispredicted branch. The number of IBS aborted fetch samples is a lower bound on the amount of unsuccessful, speculative fetch activity. It is a lower bound since the instruction data delivered by completed fetches may not be used.</td>
</tr>
<tr>
<td>IBS ITLB hit</td>
<td>The number of IBS attempted fetch samples where the fetch operation initially hit in the L1 ITLB (Instruction Translation Lookaside Buffer).</td>
</tr>
<tr>
<td>IBS L1 ITLB misses (and L2 ITLB hits)</td>
<td>The number of IBS attempted fetch samples where the fetch operation initially missed in the L1 ITLB and hit in the L2 ITLB.</td>
</tr>
<tr>
<td>IBS L1 L2 ITLB miss</td>
<td>The number of IBS attempted fetch samples where the fetch operation initially missed in both the L1 ITLB and the L2 ITLB.</td>
</tr>
<tr>
<td>IBS instruction cache misses</td>
<td>The number of IBS attempted fetch samples where the fetch operation initially missed in the IC (instruction cache).</td>
</tr>
<tr>
<td>IBS instruction cache hit</td>
<td>The number of IBS attempted fetch samples where the fetch operation initially hit in the IC.</td>
</tr>
<tr>
<td>IBS 4K page translation</td>
<td>The number of IBS attempted fetch samples where the fetch operation produced a valid physical address (i.e., address translation completed successfully) and used a 4-KByte page entry in the L1 ITLB.</td>
</tr>
<tr>
<td>IBS 2M page translation</td>
<td>The number of IBS attempted fetch samples where the fetch operation produced a valid physical address (i.e., address translation completed successfully) and used a 2-MByte page entry in the L1 ITLB.</td>
</tr>
<tr>
<td>IBS fetch latency</td>
<td>The total latency of all IBS attempted fetch samples. Divide the total IBS fetch latency by the number of IBS attempted fetch samples to obtain the average latency of the attempted fetches that were sampled.</td>
</tr>
<tr>
<td>IBS fetch L2 cache miss</td>
<td>The instruction fetch missed in the L2 Cache.</td>
</tr>
<tr>
<td>IBS ITLB refill latency</td>
<td>The number of cycles when the fetch engine is stalled for an ITLB reload for the sampled fetch. If there is no reload, the latency will be 0.</td>
</tr>
</tbody>
</table>
Table 7.3: IBS Op derived events list

<table>
<thead>
<tr>
<th>IBS Op Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All IBS op samples</td>
<td>The number of all IBS op samples that were collected. These op samples may be branch ops, resync ops, ops that perform load/store operations, or undifferentiated ops (e.g., those ops that perform arithmetic operations, logical operations, etc.). IBS collects data for retired ops. No data is collected for ops that are aborted due to pipeline flushes, etc. Thus, all sampled ops are architecturally significant and contribute to the successful forward progress of executing programs.</td>
</tr>
<tr>
<td>IBS tag-to-retire cycles</td>
<td>The total number of tag-to-retire cycles across all IBS op samples. The tag-to-retire time of an op is the number of cycles from when the op was tagged (selected for sampling) to when the op retired.</td>
</tr>
<tr>
<td>IBS completion-to-retire cycles</td>
<td>The total number of completion-to-retire cycles across all IBS op samples. The completion-to-retire time of an op is the number of cycles from when the op completed to when the op retired.</td>
</tr>
<tr>
<td>IBS branch op</td>
<td>The number of IBS retired branch op samples. A branch operation is a change in program control flow and includes unconditional and conditional branches, subroutine calls and subroutine returns. Branch ops are used to implement AMD64 branch semantics.</td>
</tr>
<tr>
<td>IBS mispredicted branch op</td>
<td>The number of IBS samples for retired branch operations that were mispredicted. This event should be used to compute the ratio of mispredicted branch operations to all branch operations.</td>
</tr>
<tr>
<td>IBS taken branch op</td>
<td>The number of IBS samples for retired branch operations that were taken branches.</td>
</tr>
<tr>
<td>IBS mispredicted taken branch op</td>
<td>The number of IBS samples for retired branch operations that were mispredicted taken branches.</td>
</tr>
<tr>
<td>IBS return op</td>
<td>The number of IBS retired branch op samples where the operation was a subroutine return. These samples are a subset of all IBS retired branch op samples.</td>
</tr>
<tr>
<td>IBS mispredicted return op</td>
<td>The number of IBS retired branch op samples where the operation was a mispredicted subroutine return. This event should be used to compute the ratio of mispredicted returns to all subroutine returns.</td>
</tr>
<tr>
<td>IBS resync op</td>
<td>The number of IBS resync op samples. A resync op is only found in certain microcoded AMD64 instructions and causes a complete pipeline flush.</td>
</tr>
<tr>
<td>IBS all load store ops</td>
<td>The number of IBS op samples for ops that perform either a load and/or store operation. An AMD64 instruction may be translated into one (“single fastpath”), two (“double fast-path”), or several (“vector path”) ops. Each op may perform a load operation, a store operation or both a load and store operation (each to the same address). Some op samples attributed to an AMD64 instruction may perform a load/store operation while other op samples attributed to the same instruction may not. Further, some branch instructions perform load/store operations. Thus, a mix of op sample types may be attributed to a single AMD64 instruction depending upon the ops that are issued from the AMD64 instruction and the op types.</td>
</tr>
<tr>
<td>IBS load ops</td>
<td>The number of IBS op samples for ops that perform a load operation.</td>
</tr>
<tr>
<td>IBS store ops</td>
<td>The number of IBS op samples for ops that perform a store operation.</td>
</tr>
<tr>
<td>IBS L1 DTLB hit</td>
<td>The number of IBS op samples where either a load or store operation initially hit in the L1 DTLB (data translation lookaside buffer).</td>
</tr>
<tr>
<td>IBS L1 DTLB misses L2 hits</td>
<td>The number of IBS op samples where either a load or store operation initially missed in the L1 DTLB and hit in the L2 DTLB.</td>
</tr>
<tr>
<td>IBS L1 and L2 DTLB misses</td>
<td>The number of IBS op samples where either a load or store operation initially missed in both the L1 DTLB and the L2 DTLB.</td>
</tr>
<tr>
<td>IBS data cache misses</td>
<td>The number of IBS op samples where either a load or store operation initially missed in the data cache (DC).</td>
</tr>
<tr>
<td>IBS data cache hits</td>
<td>The number of IBS op samples where either a load or store operation initially hit in the data cache (DC).</td>
</tr>
<tr>
<td>IBS misaligned data access</td>
<td>The number of IBS op samples where either a load or store operation caused a misaligned access (i.e., the load or store operation crossed a 128-bit boundary).</td>
</tr>
<tr>
<td>IBS bank conflict on load op</td>
<td>The number of IBS op samples where either a load or store operation caused a bank conflict with a load operation.</td>
</tr>
</tbody>
</table>

Continued on next page

7.1. CPU Performance Metrics 65
## Table 7.3 – continued from previous page

<table>
<thead>
<tr>
<th>IBS Op Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBS bank conflict on store op</td>
<td>The number of IBS op samples where either a load or store operation caused a bank conflict with a store operation.</td>
</tr>
<tr>
<td>IBS store-to-load forwarded</td>
<td>The number of IBS op samples where data for a load operation was forwarded from a store operation.</td>
</tr>
<tr>
<td>IBS store-to-load cancelled</td>
<td>The number of IBS op samples where data forwarding to a load operation from a store was cancelled.</td>
</tr>
<tr>
<td>IBS UC memory access</td>
<td>The number of IBS op samples where a load or store operation accessed uncacheable (UC) memory.</td>
</tr>
<tr>
<td>IBS WC memory access</td>
<td>The number of IBS op samples where a load or store operation accessed write combining (WC) memory.</td>
</tr>
<tr>
<td>IBS locked operation</td>
<td>The number of IBS op samples where a load or store operation was a locked operation.</td>
</tr>
<tr>
<td>IBS MAB hit</td>
<td>The number of IBS op samples where a load or store operation hit an already allocated entry in the Miss Address Buffer (MAB).</td>
</tr>
<tr>
<td>IBS L1 DTLB 4K page</td>
<td>The number of IBS op samples where a load or store operation produced a valid linear (virtual) address and a 4-KByte page entry in the L1 DTLB was used for address translation.</td>
</tr>
<tr>
<td>IBS L1 DTLB 2M page</td>
<td>The number of IBS op samples where a load or store operation produced a valid linear (virtual) address and a 2-MByte page entry in the L1 DTLB was used for address translation.</td>
</tr>
<tr>
<td>IBS L1 DTLB 1G page</td>
<td>The number of IBS op samples where a load or store operation produced a valid linear (virtual) address and a 1-GByte page entry in the L1 DTLB was used for address translation.</td>
</tr>
<tr>
<td>IBS L2 DTLB 4K page</td>
<td>The number of IBS op samples where a load or store operation produced a valid linear (virtual) address, hit the L2 DTLB, and used a 4 KByte page entry for address translation.</td>
</tr>
<tr>
<td>IBS L2 DTLB 2M page</td>
<td>The number of IBS op samples where a load or store operation produced a valid linear (virtual) address, hit the L2 DTLB, and used a 2-MByte page entry for address translation.</td>
</tr>
<tr>
<td>IBS L2 DTLB 1G page</td>
<td>The number of IBS op samples where a load or store operation produced a valid linear (virtual) address, hit the L2 DTLB, and used a 1-GByte page entry for address translation.</td>
</tr>
<tr>
<td>IBS data cache miss load latency</td>
<td>The total DC miss load latency (in processor cycles) across all IBS op samples that performed a load operation and missed in the data cache. The miss latency is the number of clock cycles from when the data cache miss was detected to when data was delivered to the core. Divide the total DC miss load latency by the number of data cache misses to obtain the average DC miss load latency.</td>
</tr>
<tr>
<td>IBS load resync</td>
<td>Load Resync.</td>
</tr>
<tr>
<td>IBS Northbridge local</td>
<td>The number of IBS op samples where a load operation was serviced from the local processor. Northbridge IBS data is only valid for load operations that miss in both the L1 data cache and the L2 data cache. If a load operation crosses a cache line boundary, then the IBS data reflects the access to the lower cache line.</td>
</tr>
<tr>
<td>IBS Northbridge remote</td>
<td>The number of IBS op samples where a load operation was serviced from a remote processor.</td>
</tr>
<tr>
<td>IBS Northbridge local L3</td>
<td>The number of IBS op samples where a load operation was serviced by the local L3 cache.</td>
</tr>
<tr>
<td>IBS Northbridge local core L1 or L2 cache</td>
<td>The number of IBS op samples where a load operation was serviced by a cache (L1 data cache or L2 cache) belonging to a local core which is a sibling of the core making the memory request.</td>
</tr>
<tr>
<td>IBS Northbridge local L1, L2, L3 cache</td>
<td>The number of IBS op samples where a load operation was serviced by a remote L1 data cache, L2 cache or L3 cache after traversing one or more coherent HyperTransport links.</td>
</tr>
<tr>
<td>IBS Northbridge local DRAM</td>
<td>The number of IBS op samples where a load operation was serviced by local system memory (local DRAM via the memory controller).</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 7.3 – continued from previous page

<table>
<thead>
<tr>
<th>IBS Op Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBS Northbridge remote DRAM</td>
<td>The number of IBS op samples where a load operation was serviced by remote system memory (after traversing one or more coherent HyperTransport links and through a remote memory controller).</td>
</tr>
<tr>
<td>IBS Northbridge local APIC MMIO Config PCI</td>
<td>The number of IBS op samples where a load operation was serviced from local MMIO, configuration or PCI space, or from the local APIC.</td>
</tr>
<tr>
<td>IBS Northbridge remote APIC MMIO Config PCI</td>
<td>The number of IBS op samples where a load operation was serviced from remote MMIO, configuration or PCI space.</td>
</tr>
<tr>
<td>IBS Northbridge cache modified state</td>
<td>The number of IBS op samples where a load operation was serviced from local or remote cache, and the cache hit state was the Modified (M) state.</td>
</tr>
<tr>
<td>IBS Northbridge cache owned state</td>
<td>The number of IBS op samples where a load operation was serviced from local or remote cache, and the cache hit state was the Owned (O) state.</td>
</tr>
<tr>
<td>IBS Northbridge local cache latency</td>
<td>The total data cache miss latency (in processor cycles) for load operations that were serviced by the local processor.</td>
</tr>
<tr>
<td>IBS Northbridge remote cache latency</td>
<td>The total data cache miss latency (in processor cycles) for load operations that were serviced by a remote processor.</td>
</tr>
</tbody>
</table>

### 7.2 Power Metrics

#### Table 7.4: Power Metrics

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy</td>
<td>RAPL Package energy</td>
<td>RAPL MSR based Package energy. Note: Available only on Ryzen, Threadripper and EPYC.</td>
</tr>
<tr>
<td>Energy</td>
<td>RAPL Core energy</td>
<td>RAPL MSR based Core energy. Note: Available only on Ryzen, Threadripper and EPYC.</td>
</tr>
<tr>
<td>Correlated Power</td>
<td>Socket Power</td>
<td>Correlated Average Socket Power for the sampling period, reported in Watts. This is an estimated consumption value based on platform activity levels. Counters are available on socket level. Note: Available only on Ryzen, Threadripper and EPYC.</td>
</tr>
<tr>
<td>Correlated Power</td>
<td>VDDCR Soc Power</td>
<td>Average Correlated VDDCR_SOC Power for the sampling period, reported in Watts. This is an estimated consumption value based on platform activity levels. Counters are available on socket level. Note: Available only on Ryzen, Threadripper and EPYC.</td>
</tr>
<tr>
<td>Correlated Power</td>
<td>Correlated VDDCR CPU Power</td>
<td>Correlated Average VDDCR_CPU_power from per-part calculations for the sampling period, reported in Watts. This is an estimated consumption value based on platform activity levels. Counters are available on socket level. Note: Available only on Ryzen, Threadripper and EPYC.</td>
</tr>
<tr>
<td>Correlated Power</td>
<td>Correlated VDDIO Mem Power</td>
<td>Correlated Average VDDIO_MEM Power for the sampling period, reported in Watts. This is an estimated consumption value based on platform activity levels. Counters are available on socket level. Note: Available only on Ryzen, Threadripper and EPYC.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlated Power</td>
<td>Correlated VDD18</td>
<td>Correlated Average VDD18 power: Power for the sampling period, reported in Watts. This is an estimated consumption value based on platform activity levels. Counters are available on socket level. Note: Available only on Ryzen, Threadripper, and EPYC.</td>
</tr>
<tr>
<td>Power</td>
<td>Total APU Power</td>
<td>Average APU Power for the sampling period, reported in Watts. This is an estimated consumption value which is calculated based on APU activity levels.</td>
</tr>
<tr>
<td>Power</td>
<td>CPU Compute Unit</td>
<td>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.</td>
</tr>
<tr>
<td>Power</td>
<td>iGPU Power</td>
<td>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.</td>
</tr>
<tr>
<td>Power</td>
<td>PCIe-Controller</td>
<td>Average PCIe-Controller Power for the sampling period, reported in Watts. This is an estimated consumption value which is calculated based on APU activity levels. The value does not include the power consumed by PCIe devices connected to the PCIe bus.</td>
</tr>
<tr>
<td>Power</td>
<td>Memory-Controller</td>
<td>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. The value does not include the power consumed by the memory DIMMs.</td>
</tr>
<tr>
<td>Power</td>
<td>Display-Controller</td>
<td>Average Display-Controller Power for the sampling period, reported in Watts. 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.</td>
</tr>
<tr>
<td>Power</td>
<td>Cumulative APU</td>
<td>The accumulated energy consumed by the APU throughout the profile session. Reported Power in Joules. Available only in the command line tool.</td>
</tr>
<tr>
<td>Power</td>
<td>Cumulative Compute</td>
<td>The accumulated energy consumed by the CPU Compute Unit throughout the profile Unit Power session. Reported in Joules. Note: Available only in the Command line interface tool.</td>
</tr>
<tr>
<td>Power</td>
<td>Cumulative iGPU Power</td>
<td>The accumulated energy consumed by the APU’s Internal GPU throughout the profile session. Reported in Joules. Note: Available only in the Command line interface tool.</td>
</tr>
<tr>
<td>Power</td>
<td>dGPU power</td>
<td>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.</td>
</tr>
<tr>
<td>Power</td>
<td>UVD Power</td>
<td>Average power consumed by the high performance Universal Video Decoder (UVD), reported in Watts. The power consumed by this component is also included in the NB Power counter.</td>
</tr>
<tr>
<td>Power</td>
<td>SMU Power</td>
<td>Average power consumed by the System Management Unit (SMU) micro controller, reported in Watts. The power consumed by this component is also included in the NB Power counter.</td>
</tr>
<tr>
<td>Power</td>
<td>RoC Power</td>
<td>Average power for the rest of the chip. This includes power consumed by parts of the APU not specifically captured by the other power counters. Reported in Watts.</td>
</tr>
</tbody>
</table>
Table 7.4 – continued from previous page

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>CPU Core/Thread Average</td>
<td>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.</td>
</tr>
<tr>
<td>Frequency</td>
<td>iGPU Average Frequency</td>
<td>Average Integrated-GPU Frequency for the sampling period, reported in MHz.</td>
</tr>
<tr>
<td>Frequency</td>
<td>dGPU Average Frequency</td>
<td>Average Discrete-GPU Frequency for the sampling period, reported in MHz. The dGPU family name is prefixed with this counter name.</td>
</tr>
<tr>
<td>Frequency</td>
<td>CPU Core/Thread Frequency</td>
<td>Histogram of CPU Core Effective Frequency (average frequency for the sampling period). Note: Available only in the Command line interface tool.</td>
</tr>
<tr>
<td>Frequency</td>
<td>iGPU Frequency Histogram</td>
<td>Histogram of Internal-GPU Effective Frequency (average frequency for the sampling period). Note: Available only in the Command line interface tool.</td>
</tr>
<tr>
<td>Temperature</td>
<td>CPU Compute-Unit Measured</td>
<td>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 the rise and decline of temperature.</td>
</tr>
<tr>
<td>Temperature</td>
<td>iGPU Measured Temperature</td>
<td>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.</td>
</tr>
<tr>
<td>Temperature</td>
<td>dGPU Measured Temperature</td>
<td>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.</td>
</tr>
<tr>
<td>P-State</td>
<td>CPU P-State</td>
<td>CPU Core P-State at the time when sampling was performed.</td>
</tr>
<tr>
<td>PPT-Limit</td>
<td>Controllers</td>
<td>Socket power limit.</td>
</tr>
<tr>
<td>PPT-Power</td>
<td>Controllers</td>
<td>Socket power filtered with the PPT time constant.</td>
</tr>
</tbody>
</table>

### 7.3 AMD Processor events reference

For the processor specific PMC events and their descriptions, refer:

**AMD Developer Documents**
- Open Source Register Reference (OSRR) for AMD Family 17h Processors
- Processor Programming Reference (PPR) for AMD Family 17h Model 00h-0Fh Processors
- Software Optimization Guide for AMD Family 17h Processors

---

7.3. AMD Processor events reference 69
INDEX

AMDTCounter (C++ enum), 54
AMDPwrCounterDesc (C++ type), 55
AMDPwrCounterValue (C++ type), 56
AMDPwrEnableCounter (C++ function), 51
AMDPwrGetCounterDesc (C++ function), 51
AMDPwrGetCounterId (C++ function), 50
AMDPwrGetSupportedCounters (C++ function), 50
AMDPwrProfileClose (C++ function), 53
AMDPwrProfileInitialize (C++ function), 49
AMDPwrProfileMode (C++ enum), 54
AMDPwrReadAllEnabledCounters (C++ function), 52
AMDPwrSample (C++ type), 55
AMDPwrSetTimerSamplingPeriod (C++ function), 52
AMDPwrStartProfiling (C++ function), 52
AMDPwrStopProfiling (C++ function), 53
AMDPwrSystemTime (C++ type), 55
AMDPwrUnit (C++ enum), 54
AMDTResult (C++ type), 54

CPUMetrics, 67

PowerMetrics, 69