AMD uProf
User Guide
The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD’s products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

**Trademarks**

AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc.

Microsoft, Windows, Windows 10 are registered trademarks of Microsoft Corporation.

MMX is a trademark of Intel Corporation.

PCIe is a registered trademark of PCI-Special Interest Group (PCI-SIG).

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.
Contents

Chapter 1  Introduction ........................................................................................................................................6
  1.1  Overview ..................................................................................................................................................6
  1.2  Specifications ........................................................................................................................................7
  1.3  Installing uProf .....................................................................................................................................8
    1.3.1 Installing Power Profiling driver on Linux .........................................................................................9
    1.3.2 Installing Remote Agent ................................................................................................................10
  1.4  Summary of Features ..............................................................................................................................10
  1.5  How does AMD uProf work? ................................................................................................................12
    1.5.1 Performance & Energy Analysis ....................................................................................................12
    1.5.2 System-wide Power Profiling .......................................................................................................14
  1.6  Support ..................................................................................................................................................14

Chapter 2  AMDuProf GUI - Quick Start ......................................................................................................15
  2.1  User Interface .......................................................................................................................................15
  2.2  Launching GUI ....................................................................................................................................17
  2.3  Configure a profile session ..................................................................................................................18
    2.3.1 Select Profile Target ..........................................................................................................................18
    2.3.2 Select Profile Type ...........................................................................................................................19
    2.3.3 Advanced Options ............................................................................................................................21
  2.4  Start Profile .........................................................................................................................................23
  2.5  Analyze the profile data .......................................................................................................................23
    2.5.1 Summary .........................................................................................................................................24
    2.5.2 Process and Function level Analysis ...............................................................................................25
    2.5.3 Source and Assembly level Analysis ...............................................................................................27
    2.5.4 Call graph .......................................................................................................................................29
    2.5.5 Flame graph ....................................................................................................................................29
    2.5.6 Instruction Mix Analysis ..................................................................................................................30
  2.6  Analyze Power profile metrics ..............................................................................................................30
  2.7  Importing Profile Databases ................................................................................................................31
  2.8  Remote Profiling ..................................................................................................................................32
  2.9  Settings ..................................................................................................................................................34
Chapter 3    Profiling with AMDuProfCLI……………………………………………….. 38

3.1   Usage ................................................................................................. 38

3.2   Options ................................................................................................. 38

3.2.1   Options for collect command ......................................................... 38

3.2.2   Options for report command ........................................................... 42

3.2.3   Options for timechart command ..................................................... 43

3.2.4   Options for info command .............................................................. 45

3.3   Performance Analysis ......................................................................... 45

3.3.1   How to Start Profiling? ................................................................. 45

3.3.2   Examples (On Windows) ................................................................. 46

3.3.3   Examples (On Linux) ..................................................................... 48

3.3.4   How to Profile Java Application? .................................................. 50

3.3.5   Linux Kernel profiling ................................................................. 51

3.3.6   System Module profiling ............................................................. 51

3.4   Energy Analysis ................................................................................. 52

3.4.1   Examples (On Windows) ................................................................. 52

3.5   System-wide Power Profiling ............................................................ 52

3.5.1   Examples (On Windows) ................................................................. 52

3.5.2   Examples (On Linux) ..................................................................... 53

Chapter 4    SDKs ......................................................................................... 54

4.1   AMDProfileControl APIs .................................................................... 54

4.1.1   Profile Control APIs ....................................................................... 54

4.1.2   Calling the APIs .............................................................................. 55

4.1.3   Compiling instrumented target application .................................. 56

4.1.4   Profiling instrumented target application ..................................... 56

4.2   AMDPowerProfileAPI Library .......................................................... 57

4.2.1   How to use API Library ................................................................. 57

Chapter 5    Limitations ............................................................................. 58

5.1   Performance Profiling ....................................................................... 58

5.2   Power Profiling .................................................................................. 58

Chapter 6    Reference ............................................................................. 59

6.1   Performance Profiling ....................................................................... 59
6.1.1 Hardware Sources ........................................................................................................... 59
6.1.2 Profiling Concepts .......................................................................................................... 60
6.1.3 Profile Types ................................................................................................................... 61
6.1.4 CPU Performance Metrics .............................................................................................. 62
6.2 Power Profiling .................................................................................................................. 70
6.2.1 Hardware Resources ....................................................................................................... 70
6.2.2 Power Metrics .................................................................................................................. 71
6.3 Useful links ......................................................................................................................... 73
Chapter 1  Introduction

1.1  Overview

AMD uProf is a performance analysis tool for applications running on Windows and Linux operating systems. It allows developers to better understand the runtime performance of their application and to identify ways to improve its performance.

AMD uProf offers:
- Performance Analysis
  - CPU Profiling - to identify runtime performance bottlenecks of the application
- Energy Analysis
  - Power Application Analysis - to identify energy hotspots in the application (Windows only)
- Power Profiling
  - System-wide Power Profiling - to monitor thermal and power characteristics of the system

AMD uProf has user interfaces:
- Graphical User Interface - AMDuProf
- Command Line Interface - AMDuProfCLI

AMD uProf can effectively be used to:
- Analyze the performance of one or more processes/applications
- Track down the performance bottlenecks in the source code
- Identify ways to optimize the source code for better performance and power efficiency
- Examine the behavior of kernel, drivers and system modules
- Observe system-level thermal and power characteristics
- Observe system metrics like IPC, memory bandwidth

Intended Audience:

This document is intended for software developers and performance tuning experts who want to improve the performance of their application. It assumes prior understanding of the concepts of threads, processes, load modules and familiarity with CPU architecture.
1.2 Specifications

AMD uProf supports the following specifications. For detailed list of supported processors and operating systems, refer Release Notes.

Processors

- AMD CPU & APU Processors
- Discrete GPUs: Graphics IP 7 GPUs, AMD Radeon 500 Series, FirePro models (Power Profiling Only)

Operating Systems

AMD uProf supports the 64-bit version of the following Operating Systems:

- Microsoft
  - Windows 7
  - Windows 10
  - Windows Server 2016
  - Windows Server 2019
- Linux
  - Ubuntu 16.04 & later
  - RHEL 7.0 & later
  - openSUSE Leap 15.0
  - SLES 12 & 15
  - CentOS 7.0 & later

Compilers and Application Environment

AMD uProf supports following application environment:

- Languages:
  - Native languages: - C, C++, Fortran, Assembly
  - Non-Native languages: - Java, C#
- Programs compiled with
  - Microsoft compilers
  - GNU compilers
  - LLVM
  - AMD’s AOCC
  - Intel compilers
• Debug info formats:
  ▪ PDB, COFF, DWARF, STABS
• Applications compiled with and without optimization or debug information
• Single-process, multi-process, single-thread, multi-threaded applications
• Dynamically linked/loaded libraries
• Programming interfaces for managed code
  ▪ Java applications - JVMTI
  ▪ CLR/.NET applications - COR
• POSIX development environment on Windows
  ▪ Cygwin
  ▪ MinGW

1.3 Installing uProf

Installer binaries are available at https://developer.amd.com/amd-uprof/. Install AMD uProf using one of the following methods.

Windows

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

Linux (using .tar.bz2 binary)

Just extract the tar.bz2 binary.

   $ tar -xf AMDuProf_Linux_x64_x.y.z.tar.bz2

To manually install the Power Profiler Linux driver, refer this section.

RHEL (using .rpm installer)

Install either using rpm or yum command.

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

Ubuntu (using .deb installer)

Install using the dpkg command.

   $ sudo dpkg --install amduprof_x.y-z_amd64.deb
1.3.1 Installing Power Profiling driver on Linux

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 the following commands:

On RHEL and CentOS distros:

$ sudo yum install gcc make

On Ubuntu distros:

$ sudo apt install build-essential       # On Debian systems

AMD uProf Debian and RPM installers perform the driver installation automatically. However, if you’ve downloaded the AMD uProf tar.bz2 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.bz2
$ 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/AMDPowerProfiler-<version number>. All the source files required for module compilation are located in this directory and are under MIT license.

To uninstall the driver run the following command:

$ cd AMDuProf_Linux_x64_x.y.z/bin
$ sudo ./AMDPowerProfilerDriver.sh uninstall

Linux Power Profiling driver support for DKMS

On Linux machines, Power profiling driver can also be installed with Dynamic Kernel Module Support (DKMS) framework support. DKMS framework automatically upgrades the power profiling driver module whenever there is a change in the existing kernel. This saves user from manually upgrading the power profiling 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 distros):

$ sudo apt-get install dkms
$ tar -xf AMDuProf_Linux_x64_x.y.z.tar.bz2
$ 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.

1.3.2 Installing Remote Agent

The AMD uProf’s remote agent **AMDRemoteAgent** is shipped with the AMD uProf installer and is installed by default when installing uProf. You can also choose to install only AMD uProf’s remote agent component while using the Windows installer.

1.4 Summary of Features

Important features of AMD uProf are:

**Performance Analysis**

AMD uProf profiler follows a statistical sampling-based approach to collect profile data to identify the performance bottlenecks in the application.

- Profile data is collected using any of the following approaches:
  - Timer Based Profiling (TBP) - to identify the hotspots in the profiled applications
  - Event Based Profiling (EBP) - sampling based on Core PMC events to identify micro-architecture related performance issues in the profiled applications
  - Instruction based Sampling (IBS)
  - Call-stack Sampling
- Scope of profile data collection:
  - Per-Process: Launch an application and profile that process its children
  - System-wide: Profile all the running processes and/or kernel
• Attach to an existing application (Native applications only)

• Profile mode
  • Profile data is collected when the application is running in User and/or Kernel mode

• Profile data is attributed at various granularities
  • Process / Thread / Load Module / Function / Source line / Disassembly
  • To correlate the profile data to Function and Source line, debug information emitted by the compiler is required
  • C++ & Java in-lined functions

• Processed profile data is stored in databases, which can be used to generate reports later.

• Profile reports are available in comma-separated-value (CSV) format to use with spreadsheets.

• **AMDuProfCLI**, the command-line-interface can be used to configure a profile run, collect the profile data and generate the profile report.
  • `collect` option to configure and collect the profile data
  • `report` option to process the profile data and to generate the profile report

• **AMDuProf** GUI can be used to:
  • Configure a profile run
  • Start the profile run to collect the performance data
  • Analyze the performance data to identify potential bottlenecks

• **AMDuProf** GUI has various UIs to analyze and view the profile data at various granularities
  • Thread concurrency graph (Windows only and requires admin privileges)
  • Source level analysis
  • Flame Graph - a stack visualizer based on collected call-stack samples
  • Call Graph - butterfly view of callgraph based on call-stack samples

• Profile Control API to selectively enable and disable profiling from the target application by instrumenting it, to limit the scope of the profiling

• Virtualized Environments -
  • Limited support for VMs
  • TBP supported on Microsoft Hyper-V, Xen, Linux KVM hypervisors
  • TBP and EBP supported in guest OS running on VMware Workstation 11.0 or later.

### System-wide Power Profiling

**AMDuProf** profiler offers live power profiling to collect thermal and power characteristics of the system.

• **AMDuProf** GUI can be used to configure and monitor the system level performance, thermal and energy metrics

• **AMDuProf** GUI’s `TIMECHART` page helps to monitor and analyze:
- Logical Core level metrics - Core Effective Frequency, P-State
- Physical Core level metrics – RAFL based Core Energy, Temperature
- Package level metrics – RAFL based Package Energy
- AMDuProfCLI’s **timechart** command to collect the system metrics and write into a text file or comma-separated-value (CSV) file
- AMDPowerProfileApi library provides APIs to configure and collect the supported system level performance, thermal and energy metrics of AMD APUs and dGPUs.
- Collected live profile data can be stored in database for future analysis

**Remote Profiling**

- Remote profiling is supported for all the profile types. The data collection will be triggered from the AMDuProf/AMDuProfCLI and the data will be collected and processed by the AMDRemoteAgent running in the target system. The AMDuProf GUI running on a host system will be used to view and analyze the profile data.
  - Host OS: Windows, Linux
  - Target OS: Windows, Linux

### 1.5 How does AMD uProf work?

#### 1.5.1 Performance & Energy Analysis

This is used to identify the performance and energy hotspots in an application. This type of profiling has the following stages:

- Collect
- Translate
- Report / Analyze

**Collect**

AMD uProf profiler follows a statistical sampling-based approach to collect the profile data. It uses variety of resources as “sampling-event” to trigger a sampling-point. Following are used as sampling-events for performance analysis.

- OS timer
- Core performance counter (PMC) events
- Instruction based sampling (IBS)
AMD uProf has many predefined configurations that can be used in this stage. A predefined configuration is a group of related sampling-events. For example, **Assess Performance** is one such configuration, which is used to get the overall assessment of performance and to find potential issues for investigation. It contains following core performance monitoring events:

- CPU Cycles not Halted
- Retired Instructions
- Retired Branch Instructions
- Retired Mispredicted Branch Instructions
- Data Cache Accesses
- Data Cache Misses
- Misaligned Accesses

User can use any of the predefined collect configuration to collect the required profile data. When a sampling-point occurs upon the expiry of the sampling-interval for a sampling-event, various profile data like Instruction Pointer, Process Id, Thread Id, Call-stack will be collected and stored in a raw profile datafile. Apart from the profile data, certain meta-data related to process/thread creation and termination, module load address, module load time and unload time too will be collected and stored.

**Translate**

Once the raw profile data is collected, the metadata and samples are processed to attribute to respective process, thread, load modules and functions. These processed data are then written into an SQLite database.

**Report / Analyze**

The AMDuProfCLI generates a user readable profile report in comma-separated-value (CSV) format from DB. The report contains various sections in which the profile data is aggregated and reported at various granularities – process, thread, load modules, function, source line, assembly.

AMDuProf GUI provides various UIs that deals with the analysis of the profile data.

- **SUMMARY** page with **Hot Spots** and **Thread Concurrency Graph** windows
- **ANALYZE** page to examine the profile data at various granularities
- **SOURCES** page to examine the data at source line and assembly level
- **Flame Graph** - a stack visualizer based on collected call-stack samples
- Flame Graph - a callstack trace visualizer to identify hot call-paths
- **Call Graph** - butterfly view of callgraph based on call-stack samples
- **IMIX** - Instruction MIX analysis view
AMD uProf has many predefined view configurations - a view configuration is a collection of interesting metrics obtained from the profile data. While analyzing the profile data, user can choose any of the available views. For example, “IPC assessment” view will report only the following data:

- CPU Cycles not Halted
- Retired Instructions
- IPC
- CPI

1.5.2 System-wide Power Profiling

This is used to analyze the energy efficiency of systems based on AMD CPUs, APUs and dGPUs. This profiling type operates in two stages namely Collect and View.

Collect

AMD uProf collects power, frequency and thermal characteristics data at regular periodic interval. The “OS Timer” is used as sampling-event and metrics from various system resources are collected at the sampling-point.

View

AMDuProf GUI provides TIMECHART page to plot the timeline graphs of the live profile data and the AMDuProfCLI reports the profile data either in text or CSV format.

1.6 Support

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

AMD uProf product page - https://developer.amd.com/amd-uprof/

AMD Developer Community forum - https://community.amd.com/community/server-gurus
Chapter 2  AMDuProf GUI - Quick Start

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

2.1 User Interface

AMDuProf GUI has various pages and each page has a number of sub windows. The pages can be navigated through the top horizontal navigation bar. When a page is selected, its sub windows will be listed in the leftmost vertical pane.

AMDuProf GUI provides various UIs to configure & start a profile run, analyze profile data and for various settings. Following UIs are available:

1. HOME page
   - Welcome - has Quick Links to start a profile run, Help Links, Links to configure a new profile and a list of recently opened profiles
   - Recent Profiles - complete list of recently opened profiles
- **Open Profile** - to import profile database
- **About** - uProf version and build information

2. **PROFILE** page
   - **Start Profiling** - window to configure a profile run by selecting profile target, profile type and start the profile
   - **Saved Configurations** - complete list of recently configured profiles
   - **Remote Profile** - to establish a connection with a remote target system

3. **SUMMARY** page
   - **Hot Spots** - overview of the hotspots for the profile session
   - **Thread Concurrency Graph** - thread concurrency graph showing number of threads running concurrently for the time elapsed
   - **Session Information** - information about this profile session
   - **System Information** - information about the target system on which profile data was collected

4. **ANALYZE** page
   - **Profile Samples** - to examine the profile data at various granularities
   - **Call Graph Samples**
     - Butterfly view of callgraph data based on call-stack samples
     - **Flame Graph** - a stack visualizer based on collected call-stack samples
   - **IMIX** - Instruction MIX analysis windows

5. **SOURCES** page
   - Per source file windows to examine the data at source line and assembly level

6. **TIMECHART** page
   - Timeline view of thermal, power, frequency and other system metrics

7. **SETTINGS** page
   - **Data Reporting** – settings that impact how the profile data is viewed
   - **Data Source** – settings that impact how data is processed – enable symbol server, specify symbol path etc.,
   - **Live Data** – settings pertaining to live timechart profile
2.2 Launching GUI

On launching the GUI, the user will be greeted with the Welcome window. This window has Quick Links to start a profile run, Help Links, Links to configure a new profile and a list of recently opened profiles.

Links in Start Here section:

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.

AMDuProf GUI’s Welcome windows

Recently Opened Profiles section will have the last 5 opened databases. Once a profile session is complete or a profile database imported, a link to that database will be added in this session.
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.

Help Links section provides links to uProf user guide and power profiler API guide.

AMD Developer Blog section provides useful links for the developers.

2.3 Configure a profile session

In order to start a profile, either click the PROFILE page at the top navigation bar or Create a new profile? link in HOME page’s Welcome window. This will navigate to the Start Profiling window.

2.3.1 Select Profile Target

You will see Select Profile Target fragment in the Start Profiling window.

Start Profiling – Select Profile Target
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:

**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).

**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.

**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.

### 2.3.2 Select 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.
Start Profiling – Select Profile Type

This fragment lets you 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.

Following profile types are available:

**CPU Profile**: This profile type is used to identify the performance bottlenecks in an application. Once this type is selected, on the left pane, the predefined sampling-event configurations will be listed. Either the user can choose any of those pre-defined events config or Custom Profile - in which the user can select the sampling-events and specify sampling-interval.

**Power App Analysis**: This profile type is used to analyze the energy consumption of an application or processes running in the system.

Start Profiling – Select Profile Type (Live Power Profile)

**System-wide Power Profile (Live)**: This profile type is used to monitor the power, thermal, frequency and other system metrics and to plot the data in a live timeline graph. Once this type is
selected, on the left pane, various supported counter categories and the components for which that category is available will be listed. The user can select the interesting counters that he wants to monitor.

2.3.3 Advanced Options

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 and related options are grouped together in each of these panes and these options are not compulsory. This is only for CPU Profile and Power App Analysis profile types.

Following panes are available to set various options for profile data collection:

Enable Thread Concurrency Option: If you want to have the thread concurrency graph, then select this option. This is Windows OS 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:**
- Call Stack Depth is Windows OS only option.
- Call Stack Options pane will not be displayed, if the selected profile target is **System.**

![Start Profiling – Advanced Options]

**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.
2.4 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:

Performance & Energy Analysis

- You will see the running timer displaying the number of seconds passed starting from zero.
- When the profiling is in progress, the user can
  - Stop the profiling by clicking Stop button.
  - Cancel the profiling by clicking Cancel button, which will take you back to Select Profile Target fragment of PROFILE
  - Pause the profiling by clicking Pause button. When the profile is paused, the profile data will not be collected, and the user can resume profiling by clicking Resume button.
- Once the profiling is stopped, the collected profile data will be processed. 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.

System-wide Power Profile (Live)

- The TIMECHART page will open and the metrics will be plotted in the live timeline graphs. Line graphs are grouped together and plotted based on the category. There is also a corresponding data table adjacent to each graph to display the current value of the counters.
- 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

2.5 Analyze the profile data

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

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

Hot Spots window

[Image of Hot Spots window]

SUMMARY – Hot Spots window

In this Hot Spots window, hotspots will be shown for functions, modules, process and threads. Process and Threads will only be shown if there are more than one. The hotspots 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 hotspot data accordingly.

The Hot Functions 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 window

Thread concurrency graph shows the number of threads of a process, running concurrently for the time elapsed (in seconds).
Apart from this, you can also see certain ancillary information regarding profile Session Information and System Information in which the profiling was carried out.

### 2.5.2 Process and Function level Analysis

Click on the ANALYZE button on the top horizontal navigation bar to go Profile Samples window, which displays the profile data table at various granularities - Process, Load Modules, Threads and Functions. The window contains data in two different formats:

- 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.
- The lower Functions 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.
ANALYZE – Profile Samples windows

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.

Filters and Options

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

- 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.
- 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.
- The Show counter values as option can be used to display the counter values either as absolute count or percentage.
- The System Modules option can be used to either exclude or include the profile data attributed to system modules.
2.5.3 Source and Assembly level Analysis

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

SOURCES – source window

Filters

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

- 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.
- The PID drop-down lists all the processes on which this selected function is executed and has samples
- The TID drop-down lists all the threads on which this selected function is executed and has samples
- The Show counter values as option can be used to display the counter values either as absolute count or percentage.
• The **Function List** drop-down list all the functions in the source file which have profile samples attributed. Selecting any one of them will take you there. However, the data for the previous one will be removed.

For multi-threaded/process applications, if a function has been executed from multiple threads/processes, then each of them will be listed in the **PID** and **TID** drop-downs in **Filters** pane. Changing them will update the counter sample data for that particular selection. By default, profile data for the selected function, aggregated across all processes and all threads will be shown.

**Source & Assembly**

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.

Note: If the source file cannot be located or opened, only disassembly will be displayed.
2.5.4 Call graph

If call-stack sampling is enabled during data collection, the Call Graph Samples window will be available in ANALYZE page to analyze the call-stack samples to identify hot call-paths. It can be navigated by clicking Call Graph Samples in the left vertical pane.

![Call Graph Samples](image)

**ANALYZE – Call graph window**

The data can be browsed based on Process IDs and Counters drop-downs. The top central table displays call-stack samples for each function. Clicking on any function updates the bottom two Caller(s) and Callee(s) tables. These tables display the callers and callees respectively of the selected function.

2.5.5 Flame graph

If call-stack sampling is enabled during data collection, the Flame Graph window will be available in Call Graph Samples window in ANALYZE page to analyze the call-stack samples to identify hot call-paths. It can be navigated by clicking Call Graph Samples in the left vertical pane and selecting the Show Flamegraph button.
The Flamegraph can be displayed based on **Process IDs** and **Counters** drop-downs. It also has the function search box to search and highlight the given function name.

### 2.5.6 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 windows** in **ANALYZE** page. In this analysis, the number of samples collected for different instruction types.

### 2.6 Analyze Power profile metrics

Once the interesting counters are selected and the profile data collection started, the **TIMECHART** page will open and the metrics will be plotted in the live timeline graphs. Line graphs are grouped together and plotted based on the category. There is also a corresponding data table adjacent to each graph to display the current value of the counters.
TIMECHART page – timeline graphs

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

2.7 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 windows and you will see the following
Similarly, in case of performance 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 windows. 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.

### 2.8 Remote Profiling

AMD uProf provides remote profiling capabilities to perform Performance and Power profiling of applications running on a remote target system. This is useful for working with headless server units.

**Adding user-id in the target system**

- Before establishing a connection with the remote agent, the user has to add the unique UID generated in the host client system. The uid can be generated either by using AMDuProfCLI or AMDuProf GUI.
To generate unique uid using AMDuProfCLI

```
C:\> AMDuProfCLI.exe info --show-uid
UID : 10976441267198678299
```

To generate unique uid using AMDuProf GUI, clicking Connect to Remote Machine? link in the Welcome window, will take you to Remote Profile window. In that window, Client ID will be displayed on top right corner. This value can be copy and pasted by selecting it right clicking.

- Add this uid to remote agent running on the remote target system

```
C:\> AMDRemoteAgent.exe -add-user 10976441267198678299
```

Running Remote Agent

The uProf remote agent **AMDRemoteAgent** runs on the remote target system allows AMD uProf clients installed on other machines to connect to that remote system and execute Performance 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 --ip 127.0.0.1
--port 20716
Local connection: IP: 127.0.0.1, port 27016
Waiting for a remote connection...
```

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 window. 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.
Remote Profile

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.

2.9 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.
SETTINGS – Data Reporting
SETTINGS – Data Source
SETTINGS – Live Data
Chapter 3  Profiling with AMDuProfCLI

AMD uProf’s command-line-interface AMDuProfCLI provides options to collect and generate report for analyzing the profile data. It can also be used from a batch file or a test script. It supports the following commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collect</td>
<td>Run the given program and collects the profile samples</td>
</tr>
<tr>
<td>report</td>
<td>Process the raw profile datafile and generates profile report</td>
</tr>
<tr>
<td>timechart</td>
<td>Power Profiling - collects and reports system characteristics like power, thermal and frequency metrics</td>
</tr>
<tr>
<td>Info</td>
<td>Displays generic information about system, topology etc.</td>
</tr>
</tbody>
</table>

3.1 Usage

Following usages are available through the supported commands:

AMDuProfCLI collect <collect-options> [PROGRAM] [ARGS]

AMDuProfCLI report <report-options>

AMDuProfCLI timechart <timechart-options> [PROGRAM] [ARGS]

AMDuProfCLI info <info-options>

In the above usage:

PROGRAM - The launch application to be profiled

ARGS - The list of arguments for the launch application

3.2 Options

3.2.1 Options for collect command

<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>
</tbody>
</table>
**collect-configs**: Predefined profile configurations that can be used with the `--config` option.

**pmu-events**: Raw PMU events that can be used with the `--event` option.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>--config</strong></td>
<td>Predefined data-collection configuration to be used to collect samples. Use the command <code>collect --list collect-configs</code> to get the list of supported configs.</td>
</tr>
<tr>
<td>**-e</td>
<td>--event**&lt; EVENT&gt;</td>
</tr>
<tr>
<td>event=&lt;timer</td>
<td>ibs-fetch</td>
</tr>
<tr>
<td>umask=&lt;unit-mask&gt;</td>
<td></td>
</tr>
<tr>
<td>user=&lt;0</td>
<td>1&gt;</td>
</tr>
<tr>
<td>os=&lt;0</td>
<td>1&gt;</td>
</tr>
<tr>
<td>interval=&lt;sampling-interval&gt;</td>
<td></td>
</tr>
<tr>
<td>ibsop-count-control=&lt;0</td>
<td>1&gt;</td>
</tr>
<tr>
<td>slicemask=&lt;L3-slice-mask&gt;</td>
<td></td>
</tr>
<tr>
<td>threadmask=&lt;L3-thread-mask&gt;</td>
<td></td>
</tr>
</tbody>
</table>

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=0xFF, threadmask=0xFF

Multiple occurrences of --event (-e) are allowed.

**NOTE:** L3 PMU events are supported only on Windows and only on Family 0x17 processors.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-p</code></td>
<td><strong>--pid &lt;PID,PID,..&gt;</strong> Profile existing processes (processes to attach to). Process IDs are separated by comma.</td>
</tr>
<tr>
<td><code>-a</code></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 -p option.</td>
</tr>
<tr>
<td><code>-c</code></td>
<td><strong>--cpu &lt;core-id,..&gt;</strong> Comma separated list of CPUs to profile. Ranges of CPUs also be specified with <code>-</code>, e.g. 0-3. Use info --cpu-topology command to get list of available core-ids.</td>
</tr>
<tr>
<td><strong>--call-graph &lt;I:D:S:F&gt;</strong></td>
<td>[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>
<tr>
<td><strong>--call-graph &lt;F:N&gt;</strong></td>
<td>[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. nofpo - 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</td>
</tr>
</tbody>
</table>
When F =nofpo, the value for N is ignored, hence no need to pass it.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-g</strong></td>
<td>[On Windows] Same as passing --call-graph 1:128:user:nofpo</td>
</tr>
<tr>
<td></td>
<td>[On Linux] Same as passing --call-graph nofpo</td>
</tr>
<tr>
<td>**-d</td>
<td>--duration &lt;n&gt;**</td>
</tr>
<tr>
<td><strong>--affinity &lt;core-id,..&gt;</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>--no-inherit</strong></td>
<td>Do not profile the children of the launched application (i.e. processes launched by the profiled application).</td>
</tr>
<tr>
<td>**-b</td>
<td>--terminate**</td>
</tr>
<tr>
<td><strong>--start-delay &lt;n&gt;</strong></td>
<td>Start Delay n in seconds. Start profiling after the specified duration. When n is 0, it has no impact.</td>
</tr>
<tr>
<td><strong>--start-paused</strong></td>
<td>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.</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>
<tr>
<td></td>
<td>1: INFO</td>
</tr>
</tbody>
</table>
### Options for report command

<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>
<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>
</tbody>
</table>
### Options for timechart command

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--list</td>
<td>Display all the supported devices and categories.</td>
</tr>
<tr>
<td>-e</td>
<td>--event &lt;type,&gt;</td>
</tr>
</tbody>
</table>

**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.

temperature: Collect all available temperature counters.

temperature: Collect all available voltage counters.

voltage: Collect all available power 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:
<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 Performance Analysis

#### 3.3.1 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.

**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 previous step.

3.3.2 Examples (On Windows)

Collect Command

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

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

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

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

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

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

- Launch AMDTClassicMatMul.exe and collect ‘TBP’ with Callstack sampling:
  
  ```
  C:\> 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):
  
  ```
  C:\> 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:
  
  ```
  C:\> AMDuProfCLI.exe collect -e event=pmcx76,interval=250000 -e event=pmcxc0,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:
  
  ```
  C:\> 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:
  
  ```
  C:\> AMDuProfCLI.exe collect -e event=timer,interval=1 -e event=l3pmcx01,umask=0x80,slicemask=0x80,threadmask=0xFF -a -d 10 -o c:\Temp\cpuprof-tb
  ```

**Report Command**

- Generate report from the raw datafile:
  
  ```
  C:\> AMDuProfCLI.exe report -i c:\Temp\cpuprof-tbp.prd -o c:\Temp\cpuprof-tbp-out
  ```

- Generate IMIX report from the raw datafile:
  
  ```
  C:\> AMDuProfCLI.exe report --imix -i c:\Temp\cpuprof-tbp.prd -o c:\Temp\cpuprof-tbp-out
  ```

- Generate report with Symbol Server paths:
  
  ```
  C:\> 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
  ```
Info command

• Print system details:
  C:\> AMDuProfCLI.exe info --system

• Print CPU topology details:
  C:\> AMDuProfCLI.exe info --cpu-topology

• To disassemble AMDTCClassicMatMul.exe into classic-disasm.txt file:
  C:\> AMDuProfCLI.exe info --disasm AMDTCClassicMatMul.exe > classic-disasm.txt

Others

• Print help in the console/terminal:
  C:\> AMDuProfCLI.exe --help

• Print uProf version:
  C:\> AMDuProfCLI.exe --version

3.3.3 Examples (On Linux)

Collect Command

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

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

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

• Collect ‘TBP’ samples in SWP mode for 10 seconds:
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

Info command

• Print system details:

  $ ./AMDuProfCLI info --system

• Print CPU topology details:

  $ ./AMDuProfCLI info --cpu-topology

• To disassemble AMDTClassicMatMul-bin into classic-disasm.txt file:

  $ ./AMDuProfCLI info --disasm AMDTClassicMatMul-bin > classic-disasm.txt
Others

- Print help in the console/terminal:
  
  ```
  $ ./AMDuProfCLI --help
  ```

- Print uProf version:
  
  ```
  $ ./AMDuProfCLI -version
  ```

### 3.3.4 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:

```
C:\> 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>.

### 3.3.5 Linux Kernel profiling

To attribute the kernel samples to appropriate functions, uProf extract required information from `/proc/kallsyms` file. Access and view of non-zero addresses from `/proc/kallsyms` file need to be provided by setting the appropriate value to `/proc/sys/kernel/kptr_restrict` file.

- When `kptr_restrict` is set to 0, kernel addresses can be viewed without any limitations.
- When `kptr_restrict` is set to 1, kernel addresses are shown if the current user has a CAP_SYSLOG capability.
- When `kptr_restrict` is set to 2, kernel addresses are hidden.

To attribute the kernel samples to kernel functions, set `/proc/sys/kernel/kptr_restrict` file to 0 before profiling.

Note:
- The address shown in `/proc/kallsyms` changes every time the system gets booted due to ASLR. To get the accurate attribution, make sure the profiling and the report generation are done in the same system powerup session.
- The settings in the `/proc/sys/kernel/kptr_restrict` file enable uProf to resolve kernel symbols and attribute samples to kernel functions. It does not enable the assembly/source level analysis, call-graph analysis.

#### Linux Kernel Module profiling

To profile a Linux kernel module, enable the settings mentioned in **Linux Kernel profiling**. Recompile the Linux kernel with the compiler option: "-g"

### 3.3.6 System Module profiling

To attribute the samples to system modules (e.g. glibc, libm, etc.), uProf uses the corresponding debug info files. Usually the Linux distros does not come with the debug info files, but most of the popular distros provide options to download the debug info files.

Refer the below links to understand how to download the debug info files.

SLES/OpenSUSE: [https://www.suse.com/support/kb/doc/?id=3074997](https://www.suse.com/support/kb/doc/?id=3074997)
Make sure to download the debug info files for the required system modules for the required Linux distros before starting the profiling.

3.4 Energy Analysis

3.4.1 Examples (On Windows)

Collect Command

- Launch classic.exe and periodically collect energy samples:
  
  ```
  C:\> AMDuProfCLI.exe collect --config power -o c:\Temp\pwrprof-swp classic.exe
  ```

- Launch classic.exe and collect ‘Power’ samples for application:
  
  ```
  C:\> AMDuProfCLI.exe collect --config power -o c:\Temp\pwrprof classic.exe
  ```

Report Command

- Generate report from the raw datafile:
  
  ```
  C:\> AMDuProfCLI.exe report -i c:\Temp\pwrprof.pdata -o c:\Temp\pwrprof-out
  ```

- Generate report with Symbol Server paths:
  
  ```
  C:\> 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
  ```

3.5 System-wide Power Profiling

3.5.1 Examples (On Windows)

- Collect all the power counter values for the duration of 10 seconds with sampling interval of 100 milliseconds:
  
  ```
  C:\> 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:
  
  ```
  C:\> AMDuProfCLI.exe timechart --event frequency -o C:\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:
C:\> AMDuProfCLI.exe timechart --event core=0-3,frequency --output C:\Temp\PowerOutput.txt --interval 500 --duration 10 --format txt

3.5.2 Examples (On Linux)

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

  $ ./AMDuProfCLI 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:

  $ ./AMDuProfCLI timechart --event frequency -o /tmp/PowerOutput.csv --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:

  $ ./AMDuProfCLI timechart --event core=0-3,frequency --output /tmp/PowerOutput.txt --interval 500 --duration 10 --format txt
Chapter 4  SDKs

AMD uProf provides following development libraries for the developers:

- Profile Control APIs - AMDProfileControl library
- APIs to collect power metrics - AMDPowerProfileAPI Library

4.1  AMDProfileControl APIs

The AMDProfileControl 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 needs to be recompiled after adding the control APIs within the application.

Header files

The application should include the header file `AMDProfileController.h` which declares the required APIs. This file is available at `include` directory under AMD uProf’s install path.

Static Library

The instrumented application should link with the `AMDProfileController` static library. This is available at:

Windows:

```plaintext
<AMDuProf-install-dir>/lib\x86\AMDProfileController.lib

<AMDuProf-install-dir>/lib\x64\AMDProfileController.lib
```

Linux:

```plaintext
<AMDuProf-install-dir>/lib/x64\libAMDProfileController.a
```

4.1.1  Profile Control APIs

These profile control APIs are available to pause and resume the profile data collection.

`amdProfileResume`
When the instrumented target application is launched through AMDuProf / AMDuProfCLI, the profiling will be in the paused state and no profile data will be collected till the application calls this resume API

```c
bool amdProfileResume (AMD_PROFILE_CPU);
```

**amdProfilePause**

When the instrumented target application wants to pause the profile data collection, this API has to be called:

```c
bool amdProfilePause (AMD_PROFILE_CPU);
```

These APIs can be called multiple times within the application. Nested Resume - Pause calls are not supported. AMD uProf profiles the code within each Resume-Pause APIs pair. After adding these APIs, the target application should be compiled before initiating a profile session.

### 4.1.2 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 Profiling data collection 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 profile data collection
    amdProfileResume (AMD_PROFILE_CPU);

    // Multiply the matrices
    multiply_matrices ();
```
4.1.3 Compiling instrumented 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 <sourcefile.cpp> -I <AMDuProf-install-dir>/include
   -L<AMDuProf-install-dir>/lib/x64/ -lAMDProfileController -lrt -pthread
```

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

4.1.4 Profiling instrumented target application

After compiling the target application, create a profile configuration 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 profiling will begin in the paused state and the target application execution begins. When the resume API gets called from target application, CPU Profile starts profiling till pause API gets called from target application or the application gets terminated. As soon as pause API is called in target application, 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
C:\> AMDuProfCLI.exe collect --config tbp --start-paused -o C:\Temp\cpuprof-tbp
C:\Program Files\AMD\AMDuProf\Examples\ClassicCpuProfileCtrl\ClassicCpuProfileCtrl.exe
```

Sample command on Linux platforms:

```bash
$ ./AMDuProfCLI collect --config tbp --start-paused -o /tmp/cpuprof-tbp
/tmp/AMDuProf/Examples/ClassicCpuProfileCtrl/ClassicCpuProfileCtrl
```
4.2  AMDPowerProfileAPI Library

AMDPowerProfileApi library provides APIs to configure and collect the supported power profiling counters on various AMD platforms. The AMDPowerProfileAPI library is used 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 AMD 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.

For detailed information on these APIs refer AMDPowerProfilerAPI.pdf

4.2.1  How to use API Library

Refer the example program CollectAllCounters.cpp on how to use these APIs. The program must be linked with AMDPowerProfileAPI library while compiling. The power profiling driver must be installed and running.

To build and execute the example application, following steps should be performed:

**Linux**

CollectAllCounters.cpp is available at <AMDuProf-install-dir>/Examples/CollectAllCounters/ dir.

```
$ cd <AMDuProf-install-dir>/Examples/CollectAllCounters

$ 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
```

To run the example program:

```
$ export LD_LIBRARY_PATH=<AMDuProf-install-dir>/bin

$ ./CollectAllCounters
```

**Windows**

A Visual Studio 2015 solution file CollectAllCounters.sln is available at /C:/Program Files/AMD/AMDuProf/Examples/CollectAllCounters/ folder to build the example program.
Chapter 5  Limitations

5.1  Performance Profiling

- CPU Profiling expects the profiled application executable binaries must not be compressed or obfuscated by any software protector tools, e.g. VMProtect.
- Thread concurrency graph is Windows only feature and requires admin privileges.
- In case of Zeppelin B1 parts, only one PMC register is used at a time for Core PMC event-based profiling (EBP).

5.2  Power Profiling

- Only one Power profile session can run at a time.
- Minimum supported sampling period in CLI is 100ms. It is recommended to use large sampling period to reduce the sampling and rendering overhead.
- 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.
- If SMU becomes in-accessible while profiling is in progress, the behavior will be undefined.
Chapter 6 Reference

6.1 Performance Profiling

The AMD uProf CPU Performance Profiling 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 Profiling uses the OS timer, HW Performance Monitor Counters (PMC), and HW IBS feature.

This section explains various key concepts related to CPU Profiling.

6.1.1 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 (L3PMC)**

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 (DFPMC)**

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

### 6.1.2 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.

- OS 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.

6.1.3 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 OS timer interval. It is used to identify the hotspots 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 several 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, OS timer, and IBS sampling events.

### 6.1.4 CPU Performance Metrics

Some of the interesting Core Performance events of AMD Processor family 17h are listed here.

**Core PMC Events**

<table>
<thead>
<tr>
<th>PMC Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[PMCx076] CPU clock cycles not halted</td>
<td>The number of core clocks that the CPU is not in a halted state.</td>
</tr>
<tr>
<td>[PMCx0C0] Retired Instructions</td>
<td>The number of instructions retired</td>
</tr>
<tr>
<td>[PMCx0C1] Retired uops</td>
<td>The number of micro-ops retired. This includes all processor activity (instructions, exceptions, interrupts, microcode assists, etc.). The number of events logged per cycle can vary from 0 to 4</td>
</tr>
<tr>
<td>[PMCx0C2] Retired Branch Instructions</td>
<td>The number of branch instructions retired. This includes all types of architectural control flow changes, including exceptions and interrupts.</td>
</tr>
<tr>
<td>[PMCx0C3] Retired Branch Instructions Mispredicted</td>
<td>The number of branch instructions retired, of any type, that were not correctly predicted in either target or direction. This includes those for which prediction is not attempted (far control transfers, exceptions and interrupts).</td>
</tr>
<tr>
<td>Event Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>[PMCx0C4] Retired Taken Branch Instructions</td>
<td>The number of taken branches that were retired. This includes all types of architectural control flow changes, including exceptions and interrupts.</td>
</tr>
<tr>
<td>[PMCx0CA] Retired Indirect Branch Instructions Mispredicted</td>
<td>Retired Indirect Branch Instructions Mispredicted</td>
</tr>
<tr>
<td>[PMCx08A] L1 BTB Correction</td>
<td>L1 BTB Correction</td>
</tr>
<tr>
<td>[PMCx08B] L2 BTB Correction</td>
<td>L2 BTB Correction</td>
</tr>
<tr>
<td>[PMCx040] Data Cache Accesses</td>
<td>The number of accesses to the data cache for load and store references. This may include certain microcode scratchpad accesses, although these are generally rare. Each increment represents an eight-byte access, although the instruction may only be accessing a portion of that. This event is a speculative event.</td>
</tr>
<tr>
<td>[PMCx041] MAB Allocation by Pipe</td>
<td>MAB allocation by pipe</td>
</tr>
<tr>
<td>[PMCx043] Data Cache Refills from System</td>
<td>Demand Data Cache Fills by Data Source</td>
</tr>
<tr>
<td>[PMCx045] L1 DTLB Miss</td>
<td>L1 DTLB Miss</td>
</tr>
<tr>
<td>[PMCx047] Misaligned loads</td>
<td>Misaligned loads (accesses).</td>
</tr>
<tr>
<td>[PMCx080] 32 Byte Instruction Cache Fetches</td>
<td>The number of 32B fetch windows transferred from IC pipe to DE instruction decoder (includes non-cacheable and cacheable fill responses)</td>
</tr>
<tr>
<td>[PMCx081] 32 Byte Instruction Cache Misses</td>
<td>The number of 32B fetch windows tried to read the L1 IC and missed in the full tag.</td>
</tr>
<tr>
<td>[PMCx084] L1 ITLB Miss, L2 ITLB Hit</td>
<td>The number of instruction-fetches that miss in the L1 ITLB but hit in the L2 ITLB</td>
</tr>
<tr>
<td>[PMCx085] L1 ITLB Miss, L2 ITLB Miss</td>
<td>The number of instruction-fetches that miss in both the L1 and L2 TLBs.</td>
</tr>
</tbody>
</table>
CPU Metrics

<table>
<thead>
<tr>
<th>CPU Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Effective Frequency</td>
<td>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.</td>
</tr>
<tr>
<td>IPC</td>
<td>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.</td>
</tr>
<tr>
<td>CPI</td>
<td>Cycles Per Instruction Retired (CPI) is the multiplicative inverse of IPC metric. This is one of the basic performance metrics indicating how cache misses, branch mis-predictions, memory latencies and other bottlenecks are affecting the execution of an application. Lower CPI value is better.</td>
</tr>
</tbody>
</table>

IBS Derived Events

AMD 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.

IBS Fetch events

<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>Event Description</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</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>
</tbody>
</table>
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.

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.

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.

The instruction fetch missed in the L2 Cache.

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.

**IBS Op events**

<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>Event Description</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</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 micro-coded 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 fast path”), 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>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>IBS L1 and L2 DTLB misses</strong></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><strong>IBS data cache misses</strong></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><strong>IBS data cache hits</strong></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><strong>IBS misaligned data access</strong></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><strong>IBS bank conflict on load op</strong></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>
<tr>
<td><strong>IBS bank conflict on store op</strong></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><strong>IBS store-to-load forwarded</strong></td>
<td>The number of IBS op samples where data for a load operation was forwarded from a store operation.</td>
</tr>
<tr>
<td><strong>IBS store-to-load cancelled</strong></td>
<td>The number of IBS op samples where data forwarding to a load operation from a store was cancelled.</td>
</tr>
<tr>
<td><strong>IBS UC memory access</strong></td>
<td>The number of IBS op samples where a load or store operation accessed uncacheable (UC) memory.</td>
</tr>
<tr>
<td><strong>IBS WC memory access</strong></td>
<td>The number of IBS op samples where a load or store operation accessed write combining (WC) memory.</td>
</tr>
<tr>
<td><strong>IBS locked operation</strong></td>
<td>The number of IBS op samples where a load or store operation was a locked operation.</td>
</tr>
<tr>
<td><strong>IBS MAB hit</strong></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><strong>IBS L1 DTLB 4K page</strong></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><strong>IBS L1 DTLB 2M page</strong></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><strong>IBS L1 DTLB 1G page</strong></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>----------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>IBS L2 DTLB 4K page</strong></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 KB page entry for address translation.</td>
</tr>
<tr>
<td><strong>IBS L2 DTLB 2M page</strong></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><strong>IBS L2 DTLB 1G page</strong></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><strong>IBS data cache miss load latency</strong></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><strong>IBS load resync</strong></td>
<td>Load Resync.</td>
</tr>
<tr>
<td><strong>IBS Northbridge local</strong></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><strong>IBS Northbridge remote</strong></td>
<td>The number of IBS op samples where a load operation was serviced from a remote processor.</td>
</tr>
<tr>
<td><strong>IBS Northbridge local L3</strong></td>
<td>The number of IBS op samples where a load operation was serviced by the local L3 cache.</td>
</tr>
<tr>
<td><strong>IBS Northbridge local core L1 or L2 cache</strong></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><strong>IBS Northbridge local core L1, L2, L3 cache</strong></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>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</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>
<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>

### 6.2 Power Profiling

#### 6.2.1 Hardware Resources

**Model Specific Registers**

- RAPL MSRs provide socket and physical core level energy consumption data. These special resisters are available on AMD family 17 processors - Ryzen, ThreadRipper and EPYC.
- Other MSRs like APERF, MPERF are used to calculate Core Effective Frequency, P-State and other profile data.

**System Management Unit (SMU)**
SMU within the AMD APU and dGPUs 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.

### 6.2.2 Power Metrics

SMU within the AMD APU and dGPUs 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.

**Supported Counter categories for Family 17h Model 00h – 0Fh (ZP)**

<table>
<thead>
<tr>
<th>Power Counter Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>Average Power for the sampling period, reported in Watts. This is an estimated consumption value based on platform activity levels. Available for Socket and VDDCR_SOC</td>
</tr>
<tr>
<td>Frequency</td>
<td>Core Effective Frequency for the sampling period, reported in MHz</td>
</tr>
<tr>
<td>Temperature</td>
<td>Average estimated temperature for the sampling period, reported in Celsius. Calculated based socket activity levels, normalized and scaled, relative to the specific processor's maximum operating temperature. Available for Die</td>
</tr>
<tr>
<td>P-State</td>
<td>CPU Core P-State at the time when sampling was performed</td>
</tr>
<tr>
<td>Energy</td>
<td>RAPL MSRs based Package and Core energy</td>
</tr>
<tr>
<td>Controllers</td>
<td>Socket PPT Limit and Power</td>
</tr>
<tr>
<td>CorrelatedPower</td>
<td>Correlated Average Power for the sampling period, reported in Watts. This is an estimated consumption value based on platform activity levels. Available for Socket, VDDR_SOC</td>
</tr>
</tbody>
</table>

**Supported Counter categories for Family 17h Model 10h – 2Fh (Raven1 and Raven2)**

<table>
<thead>
<tr>
<th>Power Counter Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>Average Power for the sampling period, reported in Watts. This is an estimated consumption value based on platform activity levels. Available for APU and VDDCR_SOC</td>
</tr>
<tr>
<td>Frequency</td>
<td>Core Effective Frequency for the sampling period, reported in MHz</td>
</tr>
</tbody>
</table>
# AMD uProf User Guide

**Temperature**

Average estimated temperature for the sampling period, reported in Celsius. Calculated based socket activity levels, normalized and scaled, relative to the specific processor's maximum operating temperature. Available for VDDCR Soc.

**P-State**

CPU Core P-State at the time when sampling was performed.

**Energy**

RAPL MSRs based Package and Core energy.

**Controllers**

Socket PPT Limit, STAPM Limit and Power.

---

## Supported Counter categories for Family 17h Model 70h – 7Fh (Matisse)

<table>
<thead>
<tr>
<th>Power Counter Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>Core Effective Frequency for the sampling period, reported in MHz</td>
</tr>
<tr>
<td>P-State</td>
<td>CPU Core P-State at the time when sampling was performed</td>
</tr>
<tr>
<td>Energy</td>
<td>RAPL MSRs based Package and Core energy</td>
</tr>
</tbody>
</table>

## Supported Counter categories for older APU families

<table>
<thead>
<tr>
<th>Power Counter Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>Average Power for the sampling period, reported in Watts. This is an estimated consumption value based on platform activity levels. Available for APU, ComputeUnit, iGPU, PCIe Controller, Memory Controller, Display Controller and VDDCR_SOC</td>
</tr>
<tr>
<td>Frequency</td>
<td>Effective Frequency for the sampling period, reported in MHz Available for Core and iGPU</td>
</tr>
<tr>
<td>Temperature</td>
<td>Average estimated temperature for the sampling period, reported in Celsius. Calculated based socket activity levels, normalized and scaled, relative to the specific processor's maximum operating temperature. Available for CPU ComputeUnit and iGPU</td>
</tr>
<tr>
<td>P-State</td>
<td>CPU Core P-State at the time when sampling was performed</td>
</tr>
<tr>
<td>Controllers</td>
<td>Socket PPT Limit and Power</td>
</tr>
</tbody>
</table>
Correlated Power

Correlated Average Power for the sampling period, reported in Watts. This is an estimated consumption value based on platform activity levels. Available for APU, CPU ComputeUnit, VDDGFX, VDDIO, VDDNB, VDDP, UVD, VCE, ACP, UNB, SMU, RoC.

Supported Counter categories for dGPUs

<table>
<thead>
<tr>
<th>Power Counter Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>Average estimated dGPU power for the sampling period, reported in Watts. Calculated based on dGPU activity levels.</td>
</tr>
<tr>
<td>Frequency</td>
<td>Average dGPU frequency for the sampling period, reported in MHz</td>
</tr>
<tr>
<td>Temperature</td>
<td>Average estimated dGPU temperature for the sampling period, reported in Celsius.</td>
</tr>
<tr>
<td>Voltage</td>
<td>CPU Core P-State at the time when sampling was performed</td>
</tr>
<tr>
<td>Current</td>
<td>Socket PPT Limit and Power</td>
</tr>
</tbody>
</table>

6.3 Useful links

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


Open Source Register Reference (OSRR) for AMD Family 17h Processors: [https://developer.amd.com/wp-content/resources/56255_3_03.PDF](https://developer.amd.com/wp-content/resources/56255_3_03.PDF)

Processor Programming Reference (PPR) for AMD Family 17h Model 00h-0Fh Processors: [http://support.amd.com/TechDocs/54945_PPR_Family_17h_Models_00h-0Fh.pdf](http://support.amd.com/TechDocs/54945_PPR_Family_17h_Models_00h-0Fh.pdf)

Software Optimization Guide for AMD Family 17h Processors: [https://developer.amd.com/wordpress/media/2013/12/55723_3_00.ZIP](https://developer.amd.com/wordpress/media/2013/12/55723_3_00.ZIP)