AMD uProf

User Guide

Advanced Micro Devices

Contents

About this document

This document describes how to use AMD uProf to perform CPU and Power analysis of applications running on Windows and Linux operating systems on AMD processors.

The latest version of this document is available at AMD uProf web site at the following URL: *https://developer.amd.com/amd-uprof/*

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 CPU architecture, concepts of threads, processes, load modules and familiarity with performance analysis concepts.

Conventions:-

Following conventions are used in this document:

Convention	**Description**
GUI element	A Graphical User Interface element like **menu name** or **button**
→	Menu item within a Menu
[]	Contents are optional in syntax
…	Preceding element can be repeated
|	Denotes "or", like two options are not allowed together
`File name`	Name of a file or path or source code snippet
`Command`	Command name or command phrase
Hyperlink	Links to external web sites
Link	Links to the section within this document

Definitions:-

Following terms may be used in this document.

Term	**Description**
PMC	Performance Monitoring Counter
TBP	Timer Based Profiling
EBP	Event Based Profiling. This uses Core PMC events.
IBS	Instruction Based Sampling
NB	Northbridge
SMU	System Management Unit
RAPL	Running Average Power Limit
MSR	Model Specific Register
DTLB	Data Translation Lookaside Buffer
DC	Data Cache
ITLB	Instruction Translation Lookaside Buffer
IC	Instruction Cache
PTI	Per Thousand Instructions
IPC	Instruction Per Cycle
CPI	Cycles Per Instruction
ASLR	Address Space Layout Randomization
GUI	Graphical User Interface
CLI	Command Line Interface
CSV	Comma Separated Values format
Target system	System in which the profile data is collected
Host system	System in which the AMDuProf client process runs

Client	Instance of AMDuProf or AMDuProfCLI running on a host system
Agent	Instance of AMDRemoteAgent process running on a target system
AMD uProf	Denotes the uProf product name
AMDuProf	Denotes the name of the graphical-user-interface tool
AMDuProfCLI	Denotes the name of the command-line-interface tool
AMDRemoteAgent	Denotes the name of the remote agent tool which runs on target system
Performance Profiling (or) CPU Profiling	Identify and analyze the performance bottlenecks. **Performance Profiling** and **CPU Profiling** denotes the same.
System Analysis	Refers the system-wide Power profiling

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

AMD uProf has following user interfaces:

- Graphical User Interface - AMDuProf
- Command Line Interface - AMDuProfCLI
- Remote Agent - AMDRemoteAgent

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

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, CentOS 7.0 & later
  - openSUSE Leap 15.0, SLES 12 & 15

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
- 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 Debian/Ubuntu distros:

```
$ sudo apt install build-essential
```

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 at `/usr/src/AMDPowerProfiler-<version>` directory. 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 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.

1.3.3 Sample program

A sample matrix multiplication application `AMDTClassicMatMul` is installed along with the product to let you use with the tool.

Windows:

```
C:\Program Files\AMD\AMDuProf\Examples\AMDTClassicMatMul\bin\AMDTClassicMatMul.exe
```

Linux:

```
/opt/AMDuProf_X.Y-ZZZ/AMDuProf/Examples/AMDTClassicMat/bin/AMDTClassicMatMul-bin
```

1.4 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 Workflow and Key concepts

2.1 Workflow

The AMD uProf workflow has the following phases:

Phase	Description
Collect	Running the application program and collect the profile data
Translate	Process the profile data to aggregate and correlate and save them in a DB
Analyze	View and analyze the performance data to identify bottlenecks

The profile data can be collected and analyzed using either by the GUI or the command-line-interface tool.

2.1.1 Collect phase

Important concepts of collect phase are explained in this section.

Profile Target

The profile target is the any of the following for which profile data will be collected.

- Application - Launch application and profile that process and its children
- System - Profile all the running processes and/or kernel
- Process - Attach to an existing application (Native applications only)

Profile Type

The profile type defines the type of profile data collected and how the data should be collected. Following profile types are supported:

- CPU Profile
- System-wide Power Profile
- Power Application Analysis (Windows only)

How data should be collected is defined by Sampling Configuration.

- **Sampling Configuration** identifies the set of Sampling Events, and their Sampling Interval and mode.
- **Sampling Event** is a resource used to trigger a sampling point at which a sample (profile data) will be collected.

- **Sampling Interval** defines the number of the occurrences of the sampling event after which an interrupt will be generated to collect the sample.
- **Mode** defines when to count the occurrences of the sampling event – in User mode and/or OS mode.

What type of profile data to collect – Sampled data:

- **Sampled data** – the profile data that can be collected when the interrupt is generated upon the expiry of the sampling interval of a sampling event.

Profile Type	Type of Profile data collected	Sampling Events
CPU Profiling	Process ID, Thread ID, IP, Callstack, ETL tracing (Windows only)	OS Timer, Core PMC events, IBS
Application Energy Analysis	Process ID, Thread ID, IP, RAPL	OS Timer
System Power Profiling	RAPL Energy values, Power & Thermal values, Core Effective Frequency	OS Timer

Sampled data

For CPU Profiling, since there are numerous micro-architecture specific events are available to monitor, the tool itself groups the related and interesting events to monitor – which is called **Predefined Sampling Configuration**. 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. Refer *this* section for all the supported Predefined Sampling Configurations.

A **Custom Sampling Configuration** is the one in which the user can define a sampling configuration with events of interest.

Profile Configuration

A profile configuration identifies all the information used to perform a collect measurement. It contains the information about profile target, sampling configuration and data to sample and profile scheduling details.

The GUI saves these profile configuration details with a default name (Ex: AMDuProf-TBP-Classic> which is also user definable. Since the performance analysis is iterative, this is persistent (can be deleted), so that the user can reuse the same configuration for future data collection runs.

Profile Session (or Profile Run)

A profile session represents a single performance experiment for a Profile Configuration. The tool saves all the profile data, translated data (in a DB) under the folder which is named as <profile config name>-<timestamp>.

Once the profile data is collected, the GUI will process the data to aggregate and attribute the samples to the respective processes, threads, load modules, functions and instructions. This aggregated data will be written into an SQLite DB which is used during Analyze phase. This process of the translating the raw profile data happens in CLI while generating the profile report.

2.1.2 Translate phase

The collected raw profile data will be processed to aggregate and attribute to the respective processes, threads, load modules, functions and instructions. Debug information for the launched application generated by the compiler is needed to correlate the samples to functions and source lines.

This phase is performed automatically in GUI once the profiling is stopped and in the CLI, when you invoke the report command to generate the report from the raw profile file.

2.1.3 Analyze phase

View Configuration

A **View** is a set of sampled event data and computed performance metrics either displayed in the GUI pages or in the text report generated by the CLI. Each predefined sampling configuration has a list of associated predefined views.

For CPU Profiling, since there are numerous micro-architecture specific events data can be collected, the tool itself groups the related and interesting metrics – which is called **Predefined View**. For example, **IPC assessment** view, lists metrics like CPU Clocks, Retired Instructions, IPC and CPI. Refer *this* section for all the supported Predefined View Configurations.

2.2 Predefined Sampling Configuration

For CPU Profiling, since there are numerous micro-architecture specific events are available to monitor, the tool itself groups the related and interesting events to monitor – which is called **Predefined Sampling Configuration**. They provide a convenient way to select a useful set of sampling events for profile analysis.

Here is the list of predefined sampling configurations:

Profile Type	Predefined Configuration Name	Abbreviation	Description
TBP	Time-based profile	tbp	To identify where programs are spending time.
EBP	Assess performance	assess	Provides an overall assessment of performance.
	Assess performance (Extended)	assess_ext	Provides an overall assessment of performance with additional metrics.
	Investigate data access	data_access	To find data access operations with poor L1 data cache locality and poor DTLB behavior.
	Investigate instruction access	inst_access	To find instruction fetches with poor L1 instruction cache locality and poor ITLB behavior.
	Investigate branching	branch	To find poorly predicted branches and near returns.
IBS	Instruction based sampling	ibs	To collect sample data using IBS Fetch and IBS OP. Precise sample attribution to instructions.
Energy	Power Application Analysis	power	To identify where the programs are consuming energy.

Note:

- The AMDuProf GUI uses the **name** of the predefined configuration in the above table.
- **Abbreviation** is used with AMDuProfCLI **collect** command's **--config** option.
- The supported predefined configurations and the sampling events used in them, is based on the processor family and model.

2.3 Predefined View Configuration

A **View** is a set of sampled event data and computed performance metrics either displayed in the GUI pages or in the text report generated by the CLI. Each predefined sampling configuration has a list of associated predefined views.

List of predefined view configurations for **Assess Performance**:

View configuration	Abbreviation	Description
Assess Performance	triage_assess	This view gives the overall picture of performance, including instructions per clock cycle (IPC), data cache accesses and misses, mispredicted branches, and misaligned data access. Use it to find possible issues for deeper investigation.
IPC assessment	ipc_assess	To find hotspots with low instruction level parallelism. Provides performance indicators – IPC and CPI.
Branch assessment	br_assess	Use this view to find code with a high branch density and poorly predicted branches.
Data access assessment	dc_assess	Information about data cache (DC) access including DC miss rate and DC miss ratio.
Misaligned access assessment	misalign_assess	To identify regions of code that access misaligned data.

List of predefined view configurations for **Investigate Data Access**:

View configuration	Abbreviation	Description
IPC assessment	ipc_assess	To find hotspots with low instruction level parallelism. Provides performance indicators – IPC and CPI.
Data access assessment	dc_assess	Information about data cache (DC) access including DC miss rate and DC miss ratio.
Data access report	dc_focus	Use this view to analyze L1 Data Cache (DC) behavior and compare misses versus refills.
Misaligned access assessment	misalign_assess	To identify regions of code that access misaligned data.
DTLB report	dtlb_focus	Information about L1 DTLB access and miss rates.

List of predefined view configurations for **Investigate Branch Access:**

View configuration	Abbreviation	Description
Investigate Branching	Branch	Use this view to find code with a high branch density and poorly predicted branches.
IPC assessment	ipc_assess	To find hotspots with low instruction level parallelism. Provides performance indicators – IPC and CPI.
Branch assessment	br_assess	Use this view to find code with a high branch density and poorly predicted branches.
Taken branch report	taken_focus	Use this view to find code with a high number of taken branches.
Near return report	return_focus	Use this view to find code with poorly predicted near returns.

List of predefined view configurations for **Assess Performance (Extended)**:

View configuration	Abbreviation	Description
Assess Performance (Extended)	triage_assess_ext	This view gives an overall picture of performance. Use it to find possible issues for deeper investigation.
IPC assessment	ipc_assess	To find hotspots with low instruction level parallelism. Provides performance indicators – IPC and CPI.
Branch assessment	br_assess	Use this view to find code with a high branch density and poorly predicted branches.
Data access assessment	dc_assess	Information about data cache (DC) access including DC miss rate and DC miss ratio.
Misaligned access assessment	misalign_assess	To identify regions of code that access misaligned data.

List of predefined view configurations for **Investigate Instruction Access:**

View configuration	Abbreviation	Description
IPC assessment	ipc_assess	To find hotspots with low instruction level parallelism. Provides performance indicators – IPC and CPI.
Instruction cache report	ic_focus	Use this view to identify regions of code that miss in the Instruction Cache (IC).
ITLB report	itlb_focus	Use this view to analyze and break out ITLB miss rates by levels L1 and L2.

List of predefined view configurations for **Instruction Based Sampling:**

View configuration	Abbreviation	Description
IBS fetch overall	ibs_fetch_overall	Use this view to show an overall summary of the IBS fetch sample data.
IBS fetch instruction cache	ibs_fetch_ic	Use this view to show a summary of IBS attempted fetch Instruction Cache (IC) miss data.
IBS fetch instruction TLB	ibs_fetch_itlb	Use this view to show a summary of IBS attempted fetch ITLB misses.
IBS fetch page translations	ibs_fetch_page	Use this view to show a summary of the IBS L1 ITLB page translations for attempted fetches.
IBS All ops	ibs_op_overall	Use this view to show a summary of all IBS Op samples.
IBS MEM all load/store	ibs_op_ls	Use this view to show a summary of IBS Op load/store data.
IBS MEM data cache	ibs_op_ls_dc	Use this view to show a summary of DC behavior derived from IBS Op load/store samples.
IBS MEM data TLB	ibs_op_ls_dtlb	Use this view to show a summary of DTLB behavior derived from IBS Op load/store data.
IBS MEM locked ops and access by type	ibs_op_ls_memacc	Use this view to show uncacheable (UC) memory access, write combining (WC) memory access and locked load/store operations.

IBS MEM translations by page size	ibs_op_ls_page	Use this view to show a summary of DTLB address translations broken out by page size.
IBS MEM forwarding and bank conflicts	ibs_op_ls_expert	Use this view to show memory access bank conflicts, data forwarding and Missed Address Buffer (MAB) hits.
IBS BR branch	ibs_op_branch	Use this view to show IBS retired branch op measurements including mispredicted and taken branches.
IBS BR return	ibs_op_return	Use this view to show IBS return op measurements including the return misprediction ratio.
IBS NB local/remote access	ibs_op_nb_access	Use this view to show the number and latency of local and remote accesses.
IBS NB cache state	ibs_op_nb_cache	Use this view to show cache owned (O) and modified (M) state for NB cache service requests.
IBS NB request breakdown	ibs_op_nb_service	Use this view to show a breakdown of NB access requests.

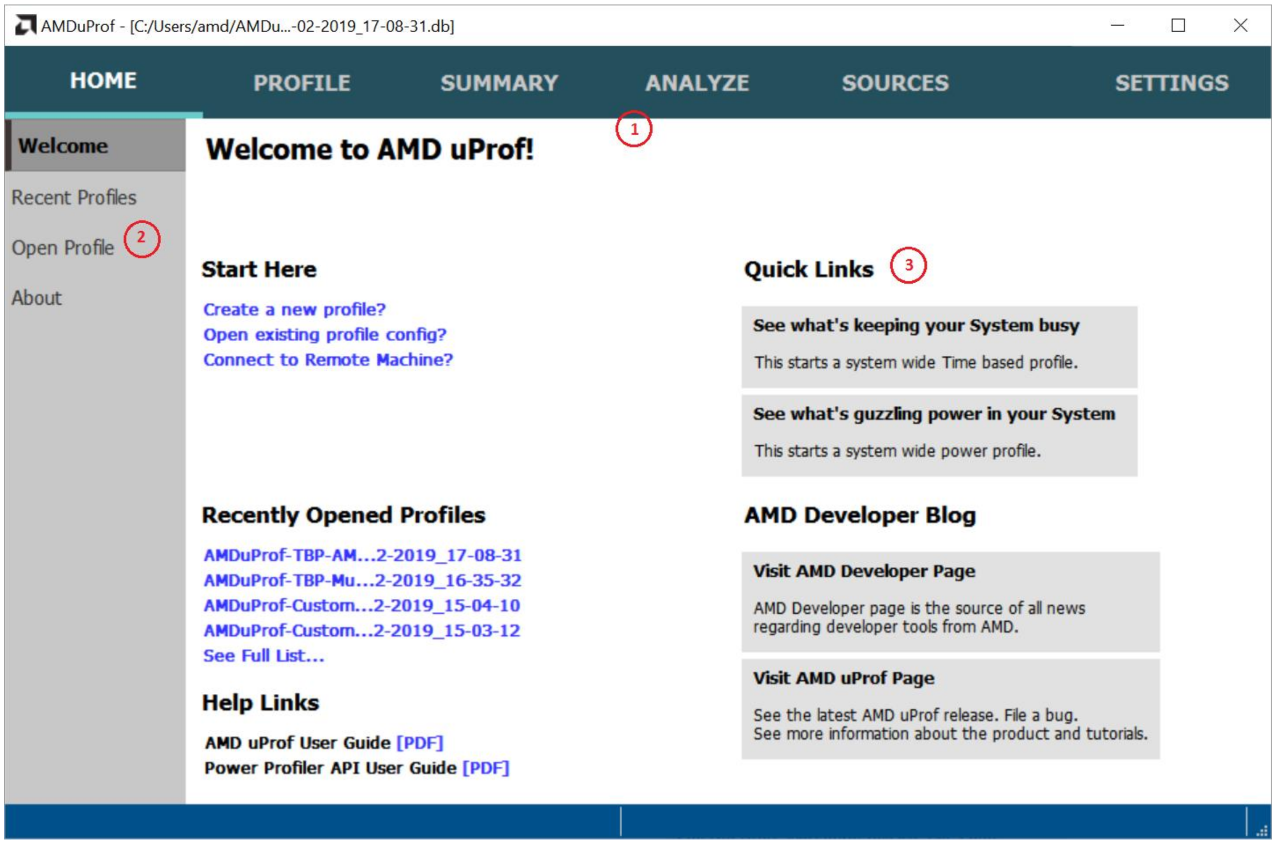
Note:

- The AMDuProf GUI uses the **name** of the predefined configuration in the above tables.
- **Abbreviation** is used with AMDuProfCLI **report** command's **--view** option.
- The supported predefined Views and the corresponding metrics are based on the processor family and model.

Chapter 3 Getting started with AMDuProf GUI

3.1 User Interface

AMDuProf GUI provides a visual interface to profile and analyze the performance data. It 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 - [C:/Users/amd/AMDu...-02-2019_17-08-31.db]
HOME
PROFILE
SUMMARY
ANALYZE
SOURCES
SETTINGS
Welcome
Recent Profiles
Open Profile
About
1
2
3
Welcome to AMD uProf!
Start Here
Create a new profile?
Open existing profile config?
Connect to Remote Machine?
Quick Links
See what's keeping your System busy
This starts a system wide Time based profile.
See what's guzzling power in your System
This starts a system wide power profile.
Recently Opened Profiles
AMDuProf-TBP-AM...2-2019_17-08-31
AMDuProf-TBP-Mu...2-2019_16-35-32
AMDuProf-Custom...2-2019_15-04-10
AMDuProf-Custom...2-2019_15-03-12
See Full List...
AMD Developer Blog
Visit AMD Developer Page
AMD Developer page is the source of all news regarding developer tools from AMD.
Visit AMD uProf Page
See the latest AMD uProf release. File a bug.
See more information about the product and tutorials.
Help Links
AMD uProf User Guide [PDF]
Power Profiler API User Guide [PDF]

AMDuProf GUI – user interface

1. The menu names in the horizontal bar like **HOME**, **PROFILE**, **SUMMARY**, **ANALYZE** are called pages
2. Each page will have its sub windows listed in the leftmost vertical pane. For example, **HOME** page has various windows like **Welcome**, **Recent Profiles**, **Open Profile** etc.,
3. Each window will have various sections. These sections are used to specify various inputs required for a profile run, display the profile data for analyze, buttons and links to navigate to

associated sections. Here in the **Welcome** window, **Quick Links** section has two links that lets you start a profile session with minimal configuration steps.

3.2 Launching GUI

To launch the AMDuProf GUI program:

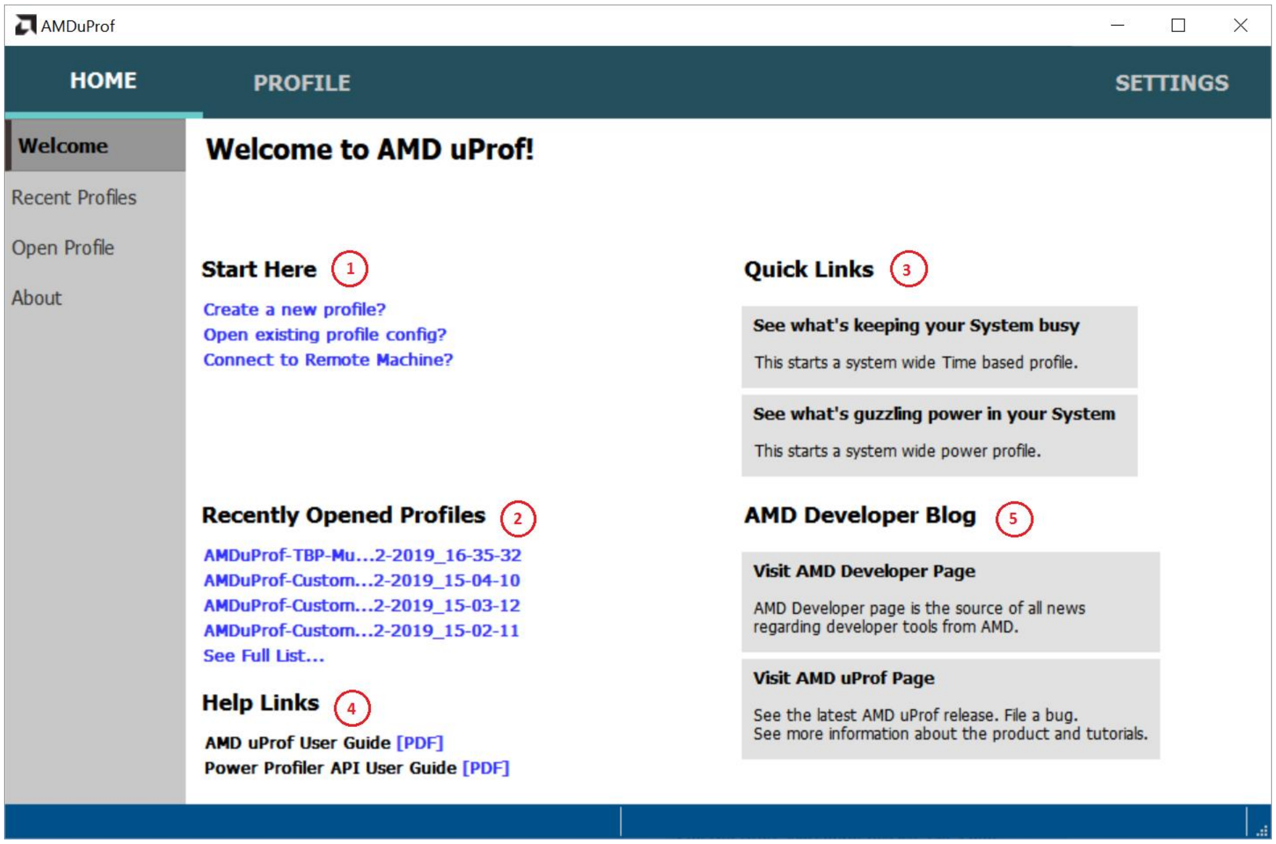
Windows

Launch GUI from `C:\Program Files\AMD\AMDuProf\bin\AMDuProf.exe` or from the Desktop shortcut.

Linux

Launch GUI from `/opt/AMDuProf_X.Y-ZZZ/AMDuProf` binary.

On launching the GUI, you will be greeted with the **Welcome** window. This window has many sections – quick links to start a profile run, help links to configure a new profile and a list of recently opened profiles.



Welcome to AMD uProf!

Start Here 1

Create a new profile?
Open existing profile config?
Connect to Remote Machine?

Recently Opened Profiles 2

AMDuProf-TBP-Mu...2-2019_16-35-32
AMDuProf-Custom...2-2019_15-04-10
AMDuProf-Custom...2-2019_15-03-12
AMDuProf-Custom...2-2019_15-02-11
See Full List...

Help Links 4

AMD uProf User Guide [PDF]
Power Profiler API User Guide [PDF]

Quick Links 3

See what's keeping your System busy
This starts a system wide Time based profile.

See what's guzzling power in your System
This starts a system wide power profile.

AMD Developer Blog 5

Visit AMD Developer Page
AMD Developer page is the source of all news regarding developer tools from AMD.

Visit AMD uProf Page
See the latest AMD uProf release. File a bug.
See more information about the product and tutorials.

AMDuProf Welcome window

1. **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.
2. **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.
3. **Quick Links** section contains two entries which lets you to start profiles with minimal configuration.
    a. Clicking **See what's keeping your System busy** will start a system-wide time-based profiling until stopped by you and then display the collected data.
    b. Clicking **See what's guzzling power in your System** will take you to a section where various power and thermal related counters can be selected and will present a live view of the data through graphs.
4. **Help Links** section provides links to uProf user guide and power profiler API guide.
5. **AMD Developer Blog** section provides useful links for the developers.

3.3 Configure a profile

To perform a collect run, first you should configure the profile by specifying the:

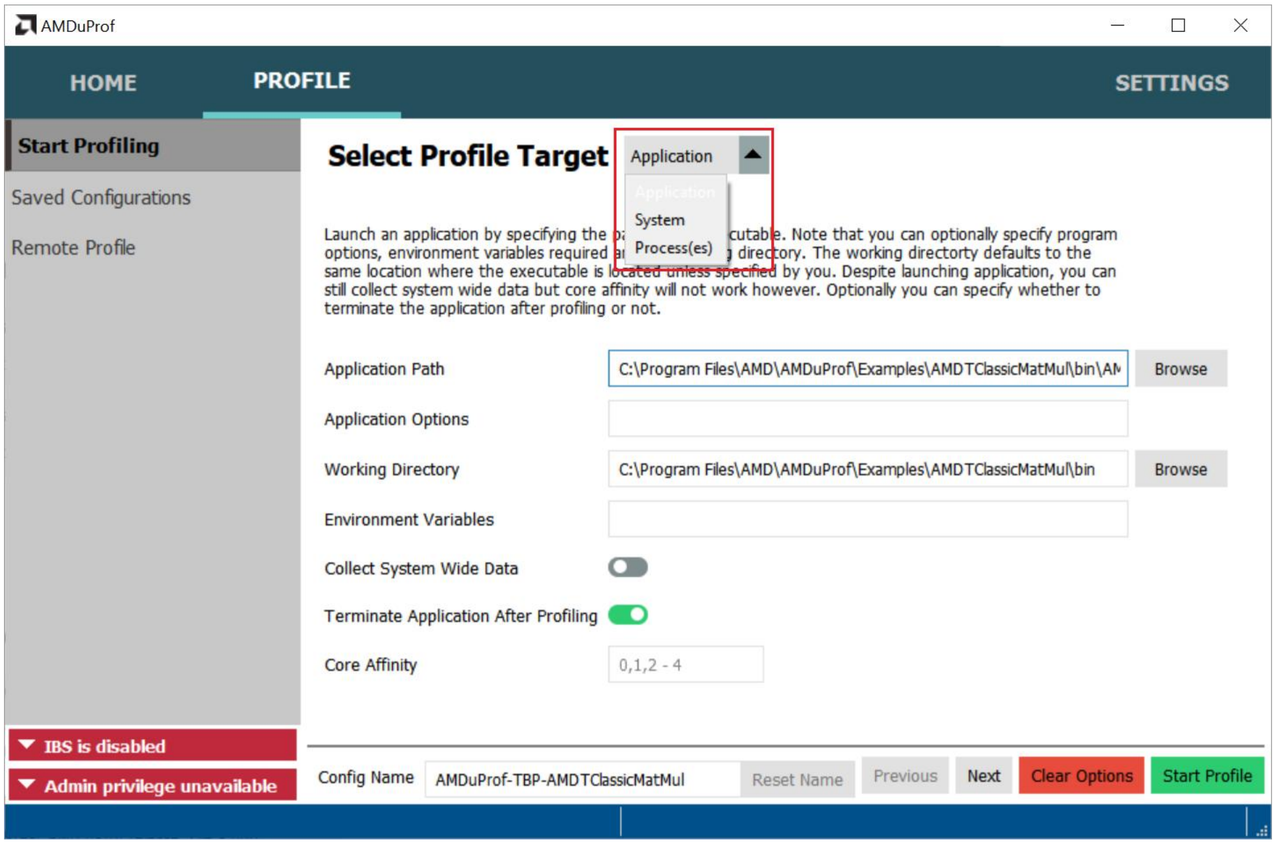
1. Profile target
2. Profile type
    a. What profile data should be collected (CPU or Power performance data)
    b. Monitoring events - how the data should be collected
    c. Additional profile data (if needed) - callstack samples, profile scheduling etc.,

This is called *profile configuration* - which identifies all the information used to perform a collect measurement. Note: The additional profile data to be collected, depends on the selected profile type.

3.3.1 Select Profile Target

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

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:



Launch an application by specifying the path to the executable. Note that you can optionally specify program options, environment variables required and working directory. The working directorty defaults to the same location where the executable is located unless specified by you. Despite launching application, you can still collect system wide data but core affinity will not work however. Optionally you can specify whether to terminate the application after profiling or not.

Application Path: C:\Program Files\AMD\AMDuProf\Examples\AMDTClassicMatMul\bin\AM

Application Options:

Working Directory: C:\Program Files\AMD\AMDuProf\Examples\AMDTClassicMatMul\bin

Environment Variables:

Collect System Wide Data

Terminate Application After Profiling

Core Affinity: 0,1,2 - 4

Config Name: AMDuProf-TBP-AMDTClassicMatMul

Start Profiling – Select Profile Target

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).

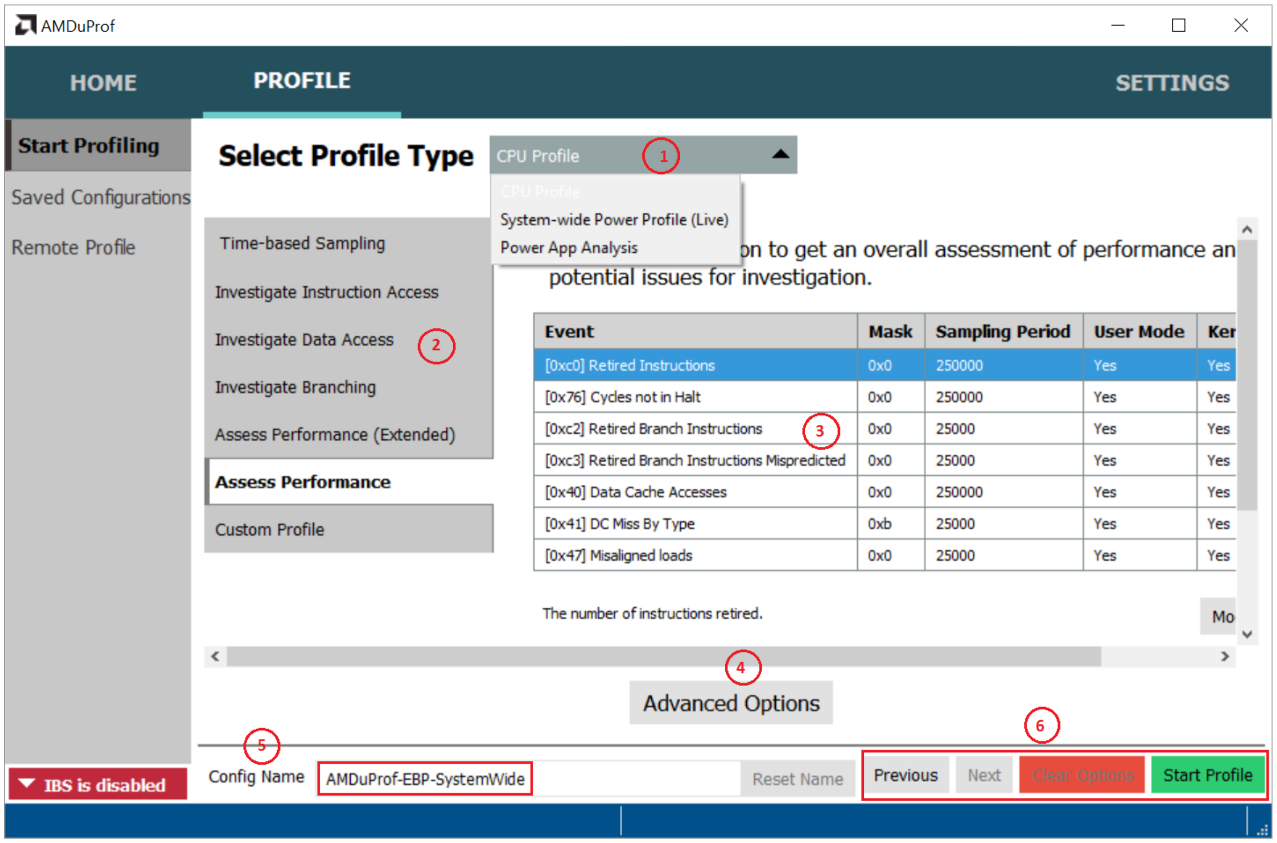
System: Select this if you do not wish to launch any application but perform either a system-wide profile or profile specific set of cores.

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.

Once profile target is selected and configured with valid data, the **Next** button will be enabled to go the next fragment of **Start Profiling**. Note that specifying any invalid option will disable the **Next** button.

3.3.2 Select Profile Type

Once profile target is selected and configured, clicking **Next** button will take you to the **Select Profile Type** fragment.



on to get an overall assessment of performance an
potential issues for investigation.

Event	Mask	Sampling Period	User Mode	Ker
[0xc0] Retired Instructions	0x0	250000	Yes	Yes
[0x76] Cycles not in Halt	0x0	250000	Yes	Yes
[0xc2] Retired Branch Instructions	0x0	25000	Yes	Yes
[0xc3] Retired Branch Instructions Mispredicted	0x0	25000	Yes	Yes
[0x40] Data Cache Accesses	0x0	250000	Yes	Yes
[0x41] DC Miss By Type	0xb	25000	Yes	Yes
[0x47] Misaligned loads	0x0	25000	Yes	Yes

The number of instructions retired.

Config Name: AMDuProf-EBP-SystemWide

Start Profiling – Select Profile Type

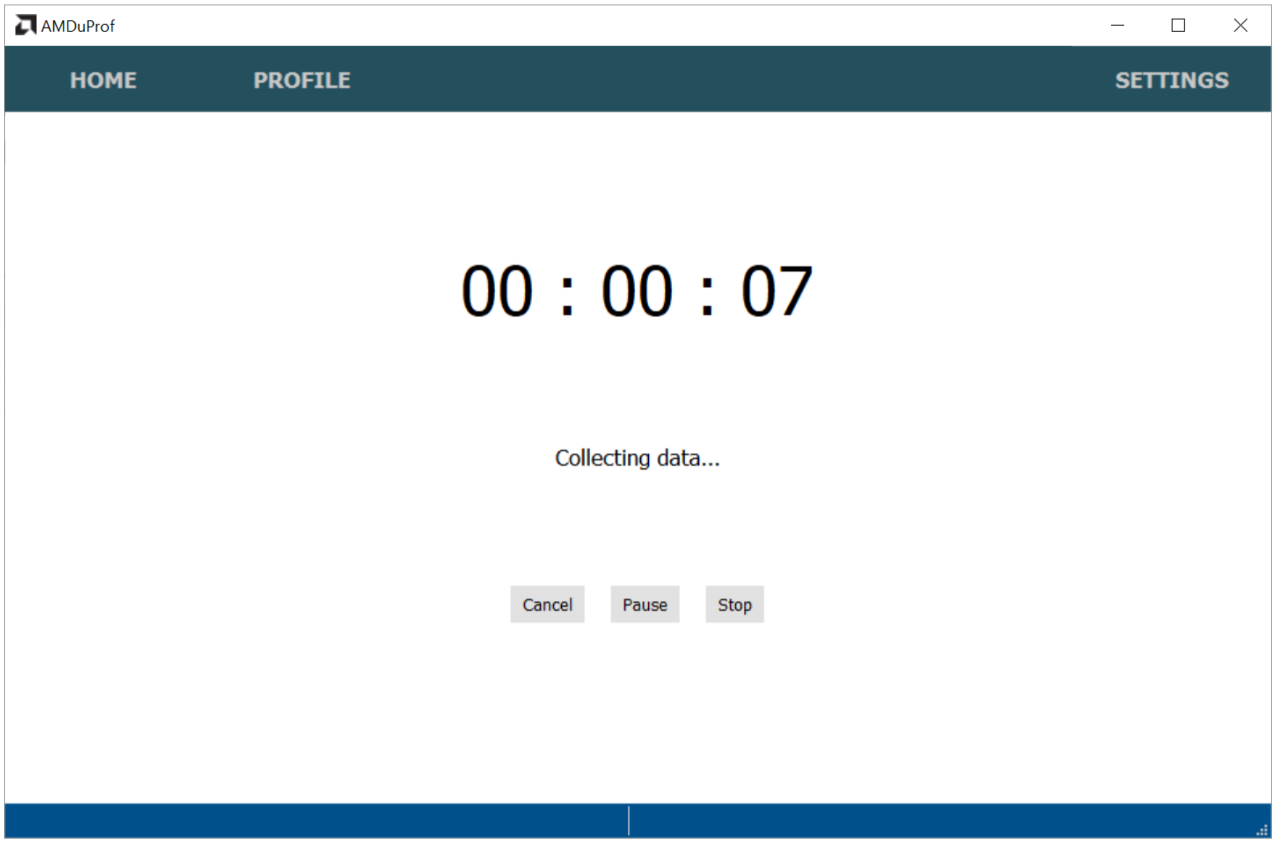
This fragment lets you to decide the type of profile data collected and how the data should be collected. You can select the profile type based on the performance analysis that you intend to perform. Refer *this* section for details on profile types. In the above figure:

1. **Select Profile Type** dropdown lists all the supported profile types
2. Once you select a profile type, the left vertical pane within this window, will list the options corresponding to the selected profile type. Here, For **CPU Profile** type, all the available predefined sampling configurations will be listed.
3. This section lists all the sampling events that are monitored in the selected predefined sampling configuration. Each entry represents a sampling configuration (Unit mask, Sampling interval, OS & User mode) for that event. You can modify these event attributes by clicking **Modify Events** button and as well add new events and/or remove events

4. Clicking **Advanced Options** button will take you to the **Advanced Options** fragment to set other options like the **Call Stack Options**, **Profile Scheduling**, **Symbols and Sources** etc.,
5. This *profile configuration* details are persistent and saved by the tool with a name – here it is AMDuProf-EBP-SystemWide. This name is user definable and the same configuration can be reused later by clinking **PROFILE → Saved Configurations** and then selecting from the list of saved configurations.
6. The Next and Previous buttons are available to navigate to various fragments within the **Start Profiling** window.

3.3.3 Start Profile

Once all the options are set correctly, the **Start Profile** button at the bottom will be enabled and you can click on it to start the profile to collect the profile data. After the profile initialization, for **CPU Profile** and **Power App Analysis** profile types, you will see:



Profile data collection

1. The running timer displaying the number of seconds passed starting from zero.
2. 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.

3.4 Analyze the profile data

When the profiling stopped, the collected raw profile data will be processed automatically, and you can analyze the profile data through various UI sections to identify the potential performance bottlenecks:

- **SUMMARY** page to look at overview of the hotspots for the profile session
- **ANALYZE** page to examine the profile data at various granularities
- **SOURCES** page to examine the data at source line and assembly level
- **TIMECHART** page to view of thermal, power, frequency and other system metrics

The sections available depends on the profile type. The **CPU Profile** and **Power Application Analysis** types will have **SUMMARY**, **ANALYZE** and **SOURCES** pages to analyze the data. The **System-wide Power Profile** will only have the **TIMECHART** page.

3.4.1 Hot Spots

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 **Session Information**.

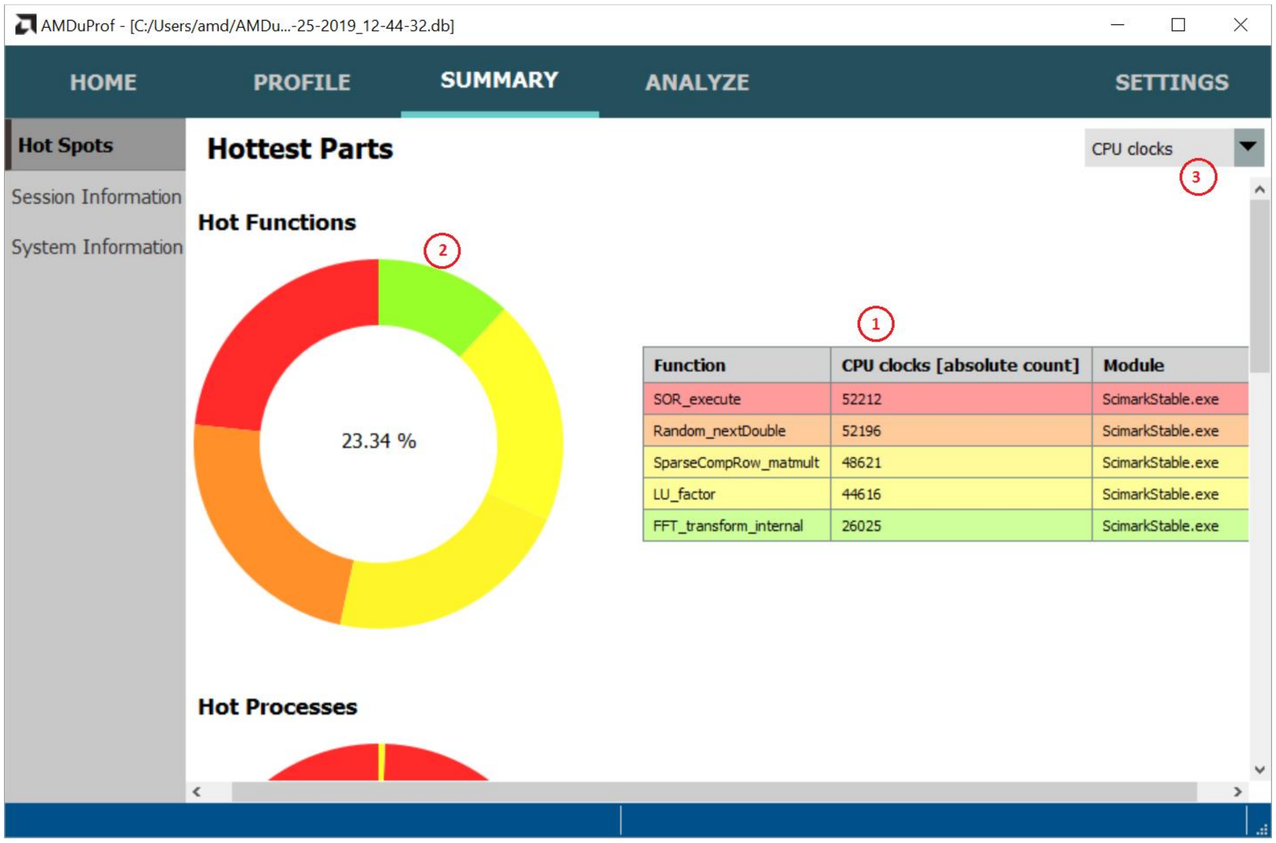
In this **Hot Spots** window, hotspots will be shown for functions, modules, process and threads. Process and Threads will only be shown if there are more than one.

In the below Hot Spots window:

1. List of top 5 hot functions for the selected event.

2. 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

3. The hotspots are shown per event and it can be selected from drop-down in top right corner. Changing it to any other event will update the hotspot data accordingly.



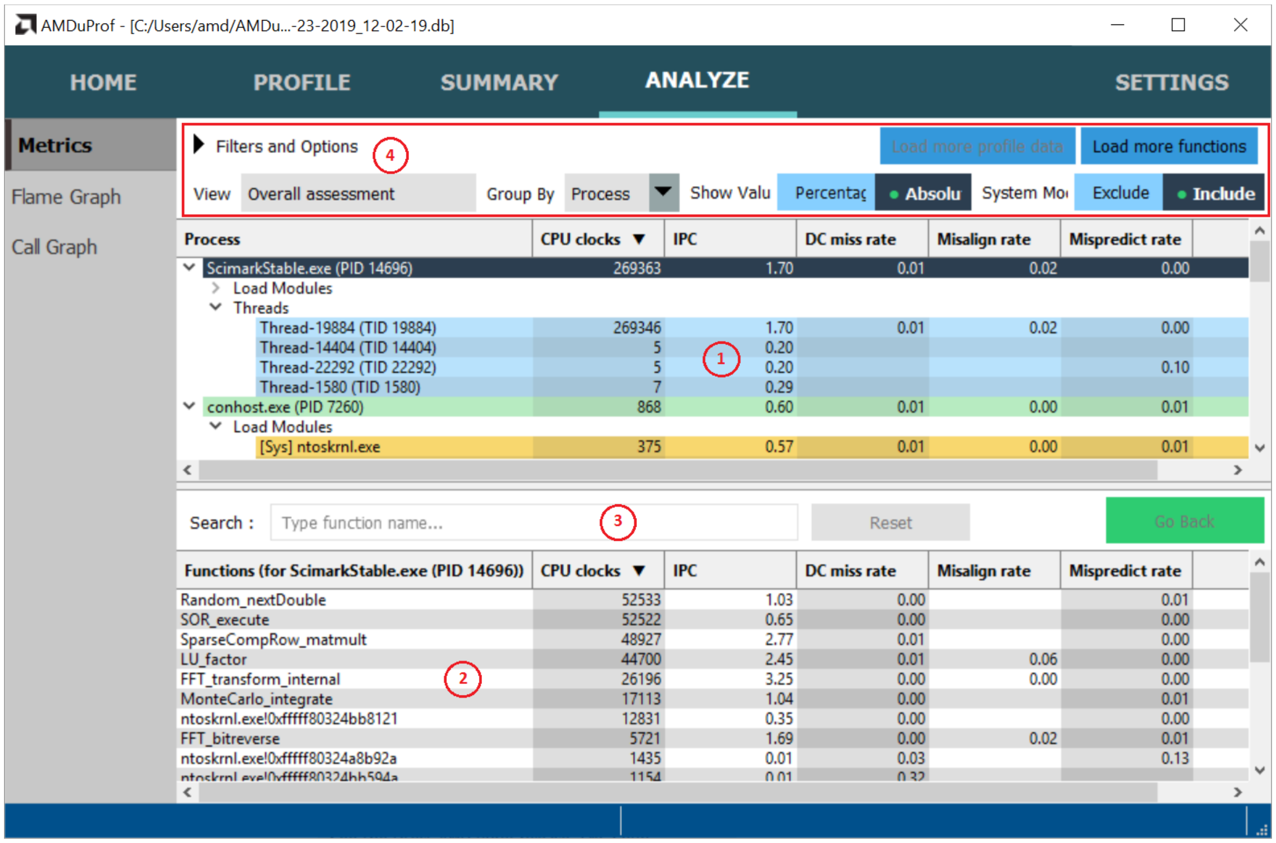
Function	CPU clocks [absolute count]	Module
SOR_execute	52212	ScimarkStable.exe
Random_nextDouble	52196	ScimarkStable.exe
SparseCompRow_matmult	48621	ScimarkStable.exe
LU_factor	44616	ScimarkStable.exe
FFT_transform_internal	26025	ScimarkStable.exe

SUMMARY – Hot Spots window

3.4.2 Process and Functions

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

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



View: Overall assessment; Group By: Process; Show Valu: Percentag / ● Absolu; System Mo: Exclude / ● Include

Process	CPU clocks ▼	IPC	DC miss rate	Misalign rate	Mispredict rate
ScimarkStable.exe (PID 14696)	269363	1.70	0.01	0.02	0.00
Load Modules					
Threads					
Thread-19884 (TID 19884)	269346	1.70	0.01	0.02	0.00
Thread-14404 (TID 14404)	5	0.20			
Thread-22292 (TID 22292)	5	0.20			0.10
Thread-1580 (TID 1580)	7	0.29			
conhost.exe (PID 7260)	868	0.60	0.01	0.00	0.01
Load Modules					
[Sys] ntoskrnl.exe	375	0.57	0.01	0.00	0.01

Search : Type function name...

Functions (for ScimarkStable.exe (PID 14696))	CPU clocks ▼	IPC	DC miss rate	Misalign rate	Mispredict rate
Random_nextDouble	52533	1.03	0.00		0.01
SOR_execute	52522	0.65	0.00		0.00
SparseCompRow_matmult	48927	2.77	0.01		0.00
LU_factor	44700	2.45	0.01	0.06	0.00
FFT_transform_internal	26196	3.25	0.00	0.00	0.00
MonteCarlo_integrate	17113	1.04	0.00		0.01
ntoskrnl.exe!0xfffff80324bb8121	12831	0.35	0.00		0.00
FFT_bitreverse	5721	1.69	0.00	0.02	0.01
ntoskrnl.exe!0xfffff80324a8b92a	1435	0.01	0.03		0.13
ntoskrnl.exe!0xfffff80324bb594a	1154	0.01	0.32		

ANALYZE page - Metrics window

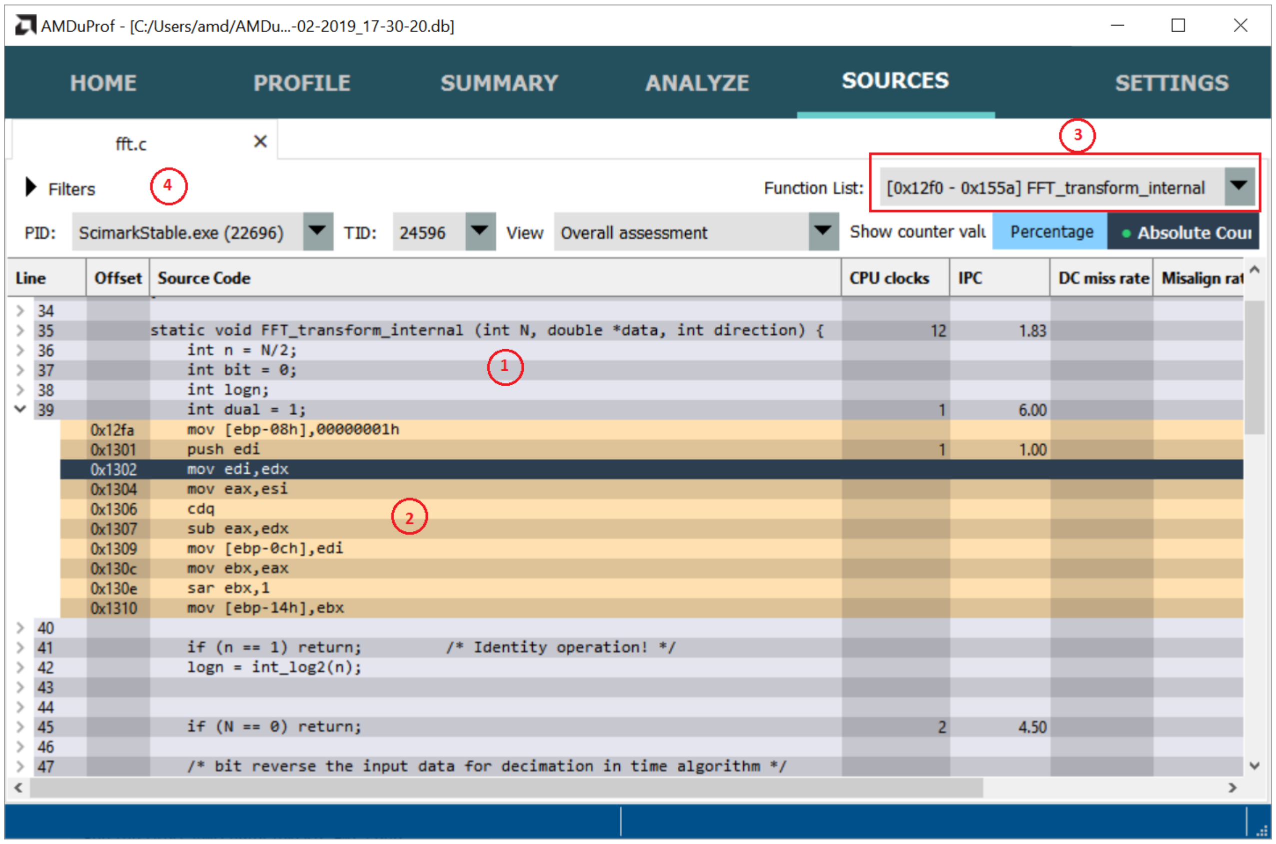
3. The search text box lets you search a function name in the **Functions** table. Only the selected function will be displayed in the Functions table. Buttons like **Reset** to clear the search text box and **Go Back** to go back to the **Functions** table that list all the functions are available
4. **Filters and Options** pane lets you filter the profile data displayed by various controls.

   - 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. Refer *this* section for more details on predefined View configurations.
   - 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 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.

Not all entries will be loaded for a profile. To load more than the default number of 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.

3.4.3 Source and Assembly

Double-clicking any entry on the **Functions** table in **Metrics** 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.



Function List: [0x12f0 - 0x155a] FFT_transform_internal

PID: ScimarkStable.exe (22696) TID: 24596 View Overall assessment Show counter valu

Line	Offset	Source Code	CPU clocks	IPC	DC miss rate	Misalign rat
34						
35		`static void FFT_transform_internal (int N, double *data, int direction) {`	12	1.83		
36		`int n = N/2;`				
37		`int bit = 0;`				
38		`int logn;`				
39		`int dual = 1;`	1	6.00		
	0x12fa	`mov [ebp-08h],00000001h`				
	0x1301	`push edi`	1	1.00		
	0x1302	`mov edi,edx`				
	0x1304	`mov eax,esi`				
	0x1306	`cdq`				
	0x1307	`sub eax,edx`				
	0x1309	`mov [ebp-0ch],edi`				
	0x130c	`mov ebx,eax`				
	0x130e	`sar ebx,1`				
	0x1310	`mov [ebp-14h],ebx`				
40						
41		`if (n == 1) return;          /* Identity operation! */`				
42		`logn = int_log2(n);`				
43						
44						
45		`if (N == 0) return;`	2	4.50		
46						
47		`/* bit reverse the input data for decimation in time algorithm */`				

SOURCES – source and assembly window

1. The source lines of the selected function are listed, and the corresponding metrics are populated in various columns against each source line. If no samples are collected when a source line was executed, the metrics column will be empty.
2. 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.
3. The **Function List** drop-down lists 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.
4. **Filters** pane lets you filter the profile data by providing the following options.

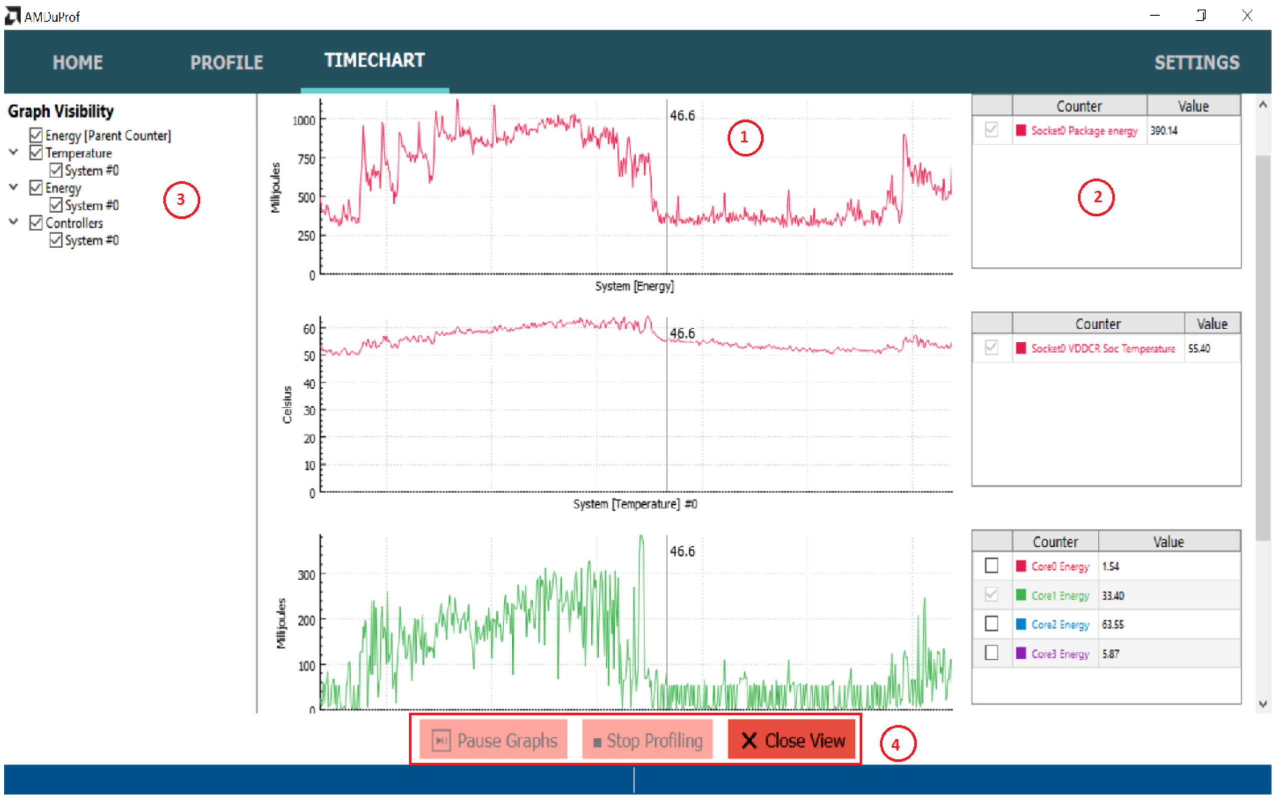
- The **View** controls the counters that are displayed. The relevant counters and their derived metrics are grouped in predefined views. The user can select it from the **View** drop-down. Refer *this* section for more details on predefined View configurations.
- The **PID** drop-down lists all the 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 Values as** option can be used to display the counter values either as absolute count or percentage.

For multi-threaded or multi-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.

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

3.4.4 Timechart

In **System-wide Power Profile(Live)** type, once the interesting counters are selected and profiling started, the **TIMECHART** page will open and the metrics will be plotted in the timeline graphs.



Graph Visibility
- ☑ Energy [Parent Counter]
- ☑ Temperature
  - ☑ System #0
- ☑ Energy
  - ☑ System #0
- ☑ Controllers
  - ☑ System #0

	Counter	Value
✓	🟥 Socket0 Package energy	390.14

	Counter	Value
✓	🟥 Socket0 VDDCR Soc Temperature	55.40

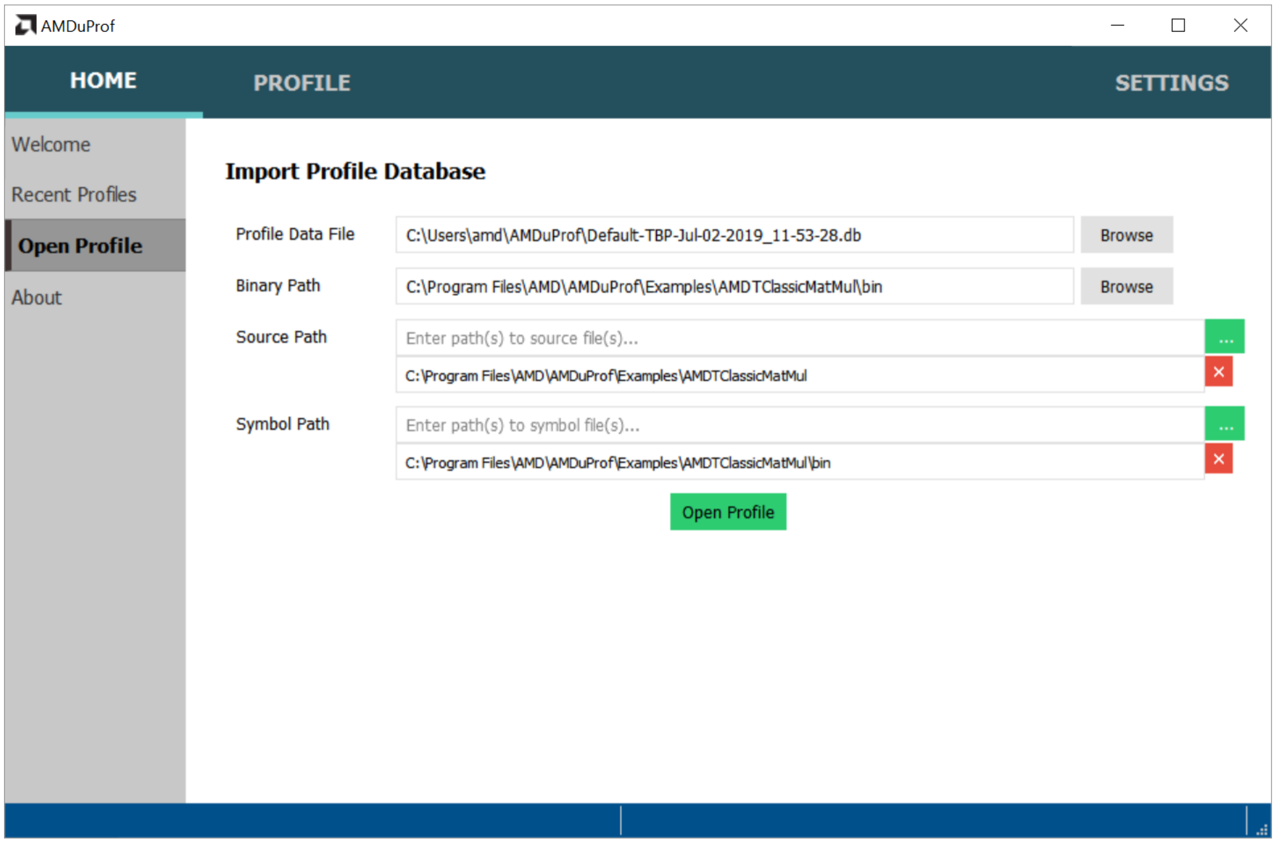
	Counter	Value
☐	🟥 Core0 Energy	1.54
✓	🟩 Core1 Energy	33.40
☐	🟦 Core2 Energy	63.55
☐	🟪 Core3 Energy	5.87

TIMECHART page – timeline graphs

1. In the **TIMECHART** page the metrics will be plotted in the timeline graphs. Line graphs are grouped together and plotted based on the category.
2. There is also a corresponding data table adjacent to each graph to display the current value of the counters.
3. Graph Visibility pane on the left vertical pane will let you choose the graph to display.
4. When plotting is in progress various buttons are available, to let you
   - 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

3.5 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** window and you will see the following window.



Import Profile Database

Profile Data File: C:\Users\amd\AMDuProf\Default-TBP-Jul-02-2019_11-53-28.db

Binary Path: C:\Program Files\AMD\AMDuProf\Examples\AMDTClassicMatMul\bin

Source Path: Enter path(s) to source file(s)...
C:\Program Files\AMD\AMDuProf\Examples\AMDTClassicMatMul

Symbol Path: Enter path(s) to symbol file(s)...
C:\Program Files\AMD\AMDuProf\Examples\AMDTClassicMatMul\bin

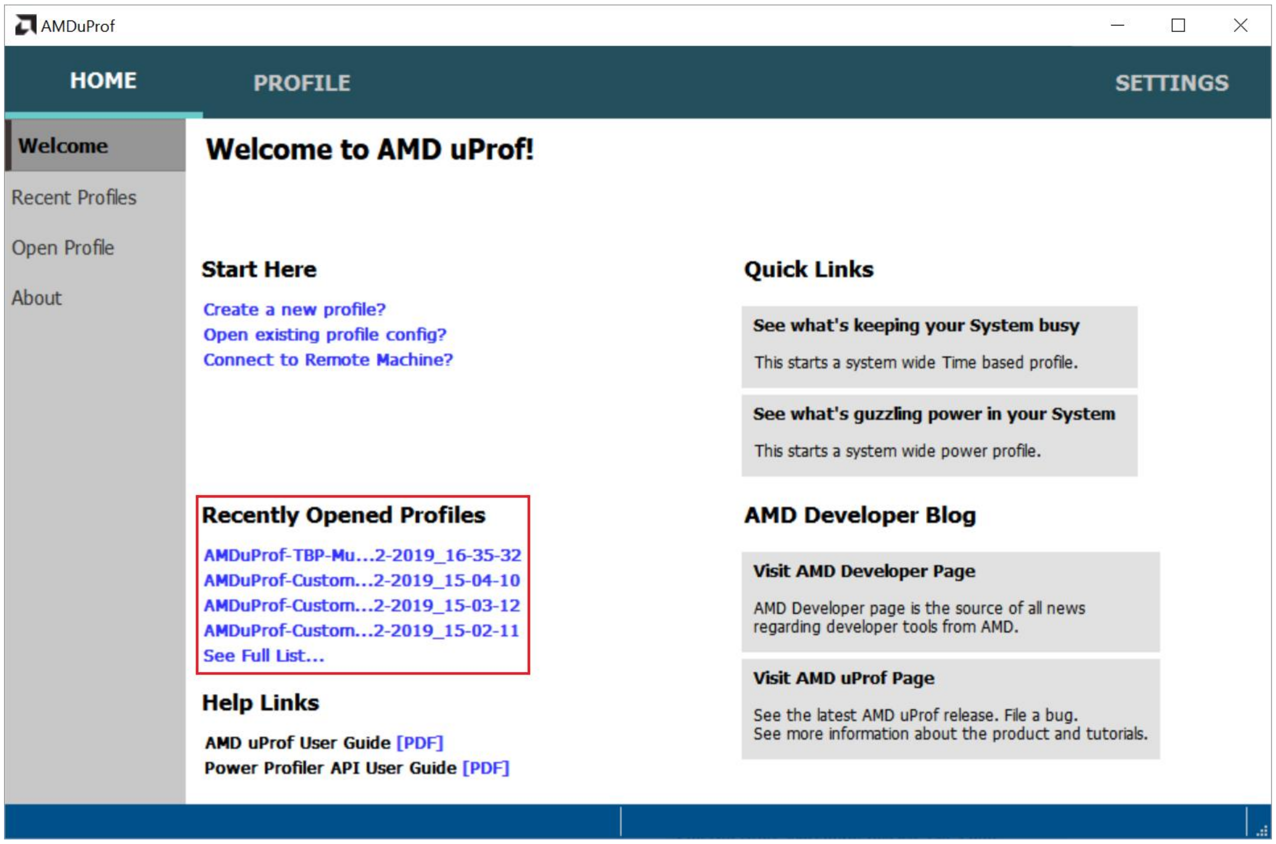
Open Profile – importing profile database

This can be used to import a raw profile data file collected using the CLI or the processed data saved in the DB as well.

- The path should be specified in the Profile Data File input text box.
- **Binary Path**: If the profile run is performed in a system and the corresponding raw profile data is imported in another system, then you may need to specify the path(s) in which binary files can be located.
- **Source Path**: Specify the source path(s) from where the sources files can be located.
- **Symbol Path**: Specify the symbol path(s) from where the debug info files (On Windows, PDB files) can be located.

3.6 Analyzing saved Profile Session

Once you have a created new profile session or opened(imported) profile database, a history is created, and the last 50 opened profile databases' records are stored (i.e. where they are located). Such a list will come up in the **HOME → Welcome** window's **Recently Opened Profiles** as well.



Welcome to AMD uProf!

Start Here

Create a new profile?
Open existing profile config?
Connect to Remote Machine?

Quick Links

See what's keeping your System busy
This starts a system wide Time based profile.

See what's guzzling power in your System
This starts a system wide power profile.

Recently Opened Profiles

AMDuProf-TBP-Mu...2-2019_16-35-32
AMDuProf-Custom...2-2019_15-04-10
AMDuProf-Custom...2-2019_15-03-12
AMDuProf-Custom...2-2019_15-02-11
See Full List...

AMD Developer Blog

Visit AMD Developer Page
AMD Developer page is the source of all news regarding developer tools from AMD.

Visit AMD uProf Page
See the latest AMD uProf release. File a bug.
See more information about the product and tutorials.

Help Links

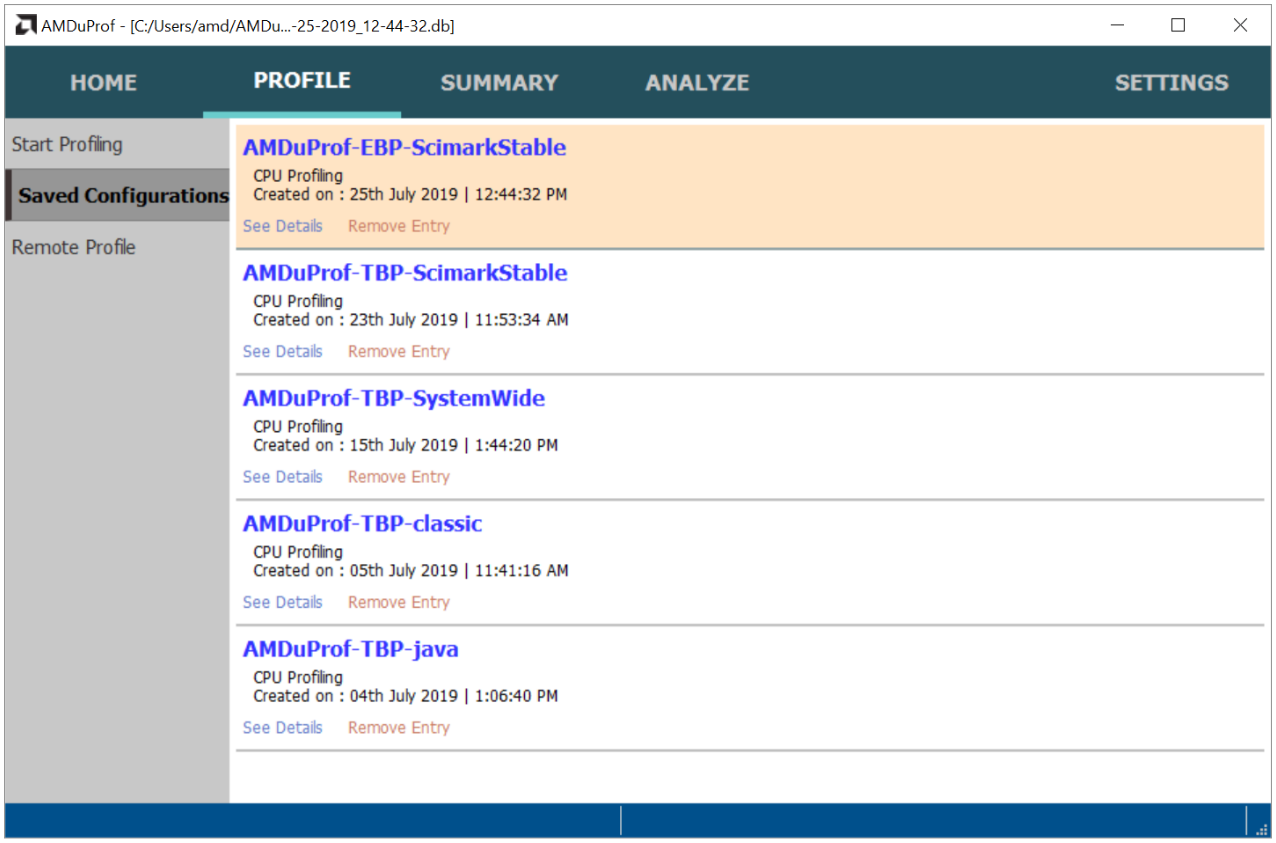
AMD uProf User Guide [PDF]
Power Profiler API User Guide [PDF]

Welcome page - Recently Opened Profile section

If there are more than 5 entries, you can open the **Recent Profiles** in **HOME** page, and you will see the list of all profile databases opened. Clicking any of the profile session entry, will load the profile data for analyzing it.

3.7 Using saved Profile Configuration

When a profile configuration is created (when you set the options and start profiling), if it generates at least one valid profile session, the profile configuration details 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** window.



AMDuProf-EBP-ScimarkStable
CPU Profiling
Created on : 25th July 2019 | 12:44:32 PM

AMDuProf-TBP-ScimarkStable
CPU Profiling
Created on : 23th July 2019 | 11:53:34 AM

AMDuProf-TBP-SystemWide
CPU Profiling
Created on : 15th July 2019 | 1:44:20 PM

AMDuProf-TBP-classic
CPU Profiling
Created on : 05th July 2019 | 11:41:16 AM

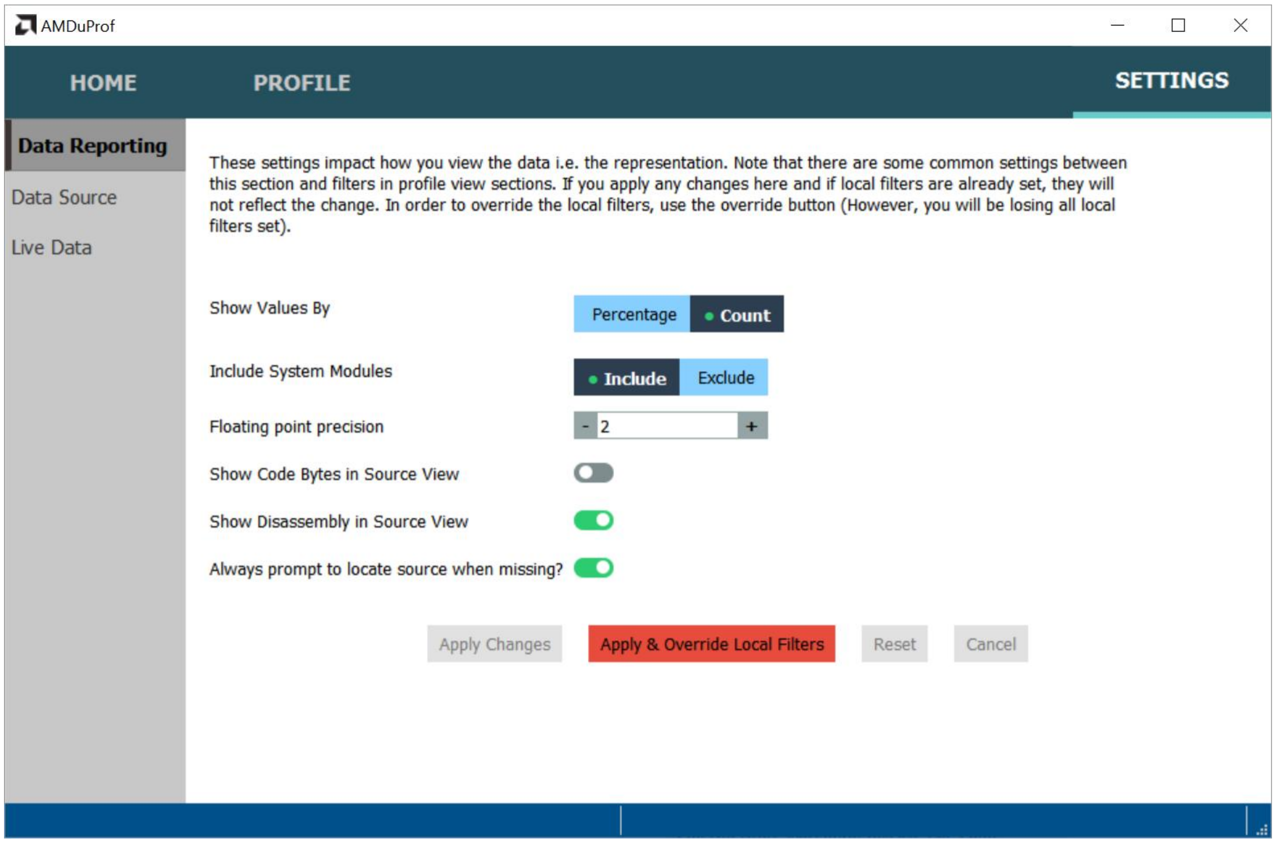
AMDuProf-TBP-java
CPU Profiling
Created on : 04th July 2019 | 1:06:40 PM

Saved Configurations

Note that by default the profile configuration 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.

3.8 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.



These settings impact how you view the data i.e. the representation. Note that there are some common settings between this section and filters in profile view sections. If you apply any changes here and if local filters are already set, they will not reflect the change. In order to override the local filters, use the override button (However, you will be losing all local filters set).

Show Values By: Percentage [✓ Count]

Include System Modules: [✓ Include] Exclude

Floating point precision: 2

Show Code Bytes in Source View

Show Disassembly in Source View

Always prompt to locate source when missing?

SETTINGS – Data Reporting

- 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. 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.

Chapter 4 Getting started with AMDuProfCLI

AMD uProf's command-line-interface AMDuProfCLI provides options to collect and generate report for analyzing the profile data.

```
AMDuProfCLI [--version] [--help] COMMAND [<options>] [<PROGRAM>] [<ARGS>]
```

Following COMMANDs are supported:

Command	Description
Collect	Run the given program and collects the profile samples
Report	Process the raw profile datafile and generates profile report
Timechart	Power Profiling - collects and reports system characteristics like power, thermal and frequency metrics
Info	Displays generic information about system, topology

Refer *this* section for the workflow. To run the command line interface AMDuProfCLI:

Windows:

Run `C:\Program Files\AMD\AMDuProf\bin\AMDuProfCLI.exe` binary.

Linux:

Run `/opt/AMDuProf_X.Y-ZZZ/AMDuProfCLI` binary.

4.1 How to start CPU profile?

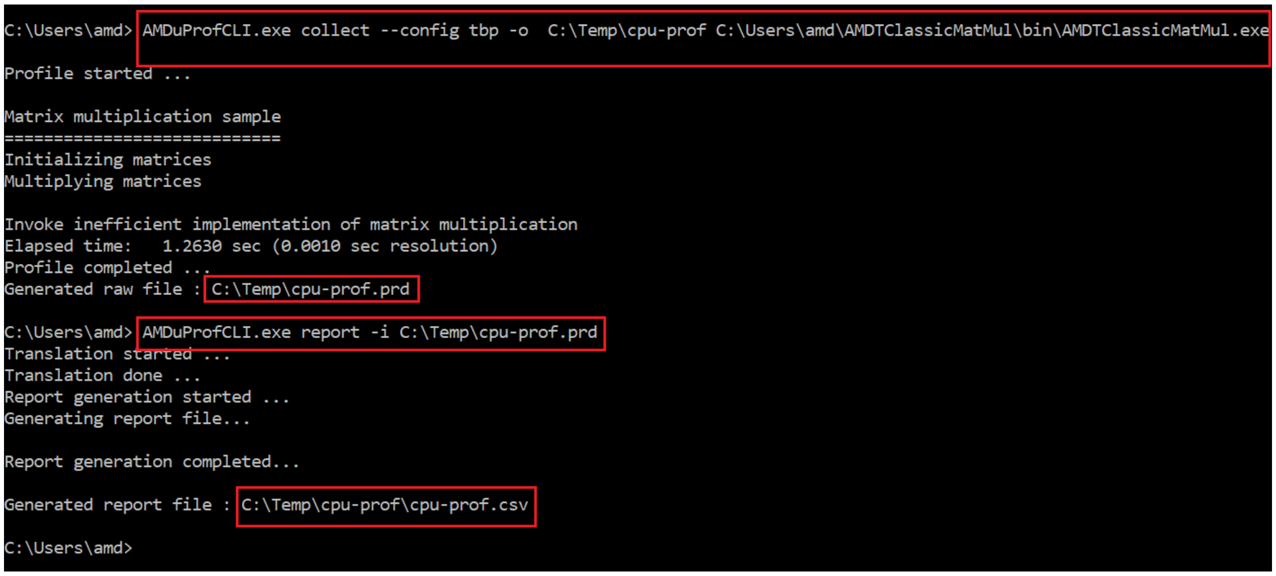
To profile and analyze the performance of a native (C/C++) application, you need to follow these steps:

1. Prepare the application. Refer *section* on how to prepare an application for profiling
2. Collect the samples for the application using AMDuProfCLI's **collect** command
3. Generate the report using AMDuProfCLI's **report** command, in readable format for analysis

Preparing the application is to build the launch application with debug information as debug info is needed to correlate the samples to functions and source lines.

The collect command will launch the application (if given) and collect the profile data for the given profile type and sampling configuration. It will generate raw data file (**.PRD** on Windows and **.caperf** on Linux) and other miscellaneous files.

The report command translates the collected raw profile data to aggregate and attribute to the respective processes, threads, load modules, functions and instructions and writes them into a DB and then generate a report in CSV format.



```
C:\Users\amd> AMDuProfCLI.exe collect --config tbp -o  C:\Temp\cpu-prof C:\Users\amd\AMDTClassicMatMul\bin\AMDTClassicMatMul.exe

Profile started ...

Matrix multiplication sample
============================
Initializing matrices
Multiplying matrices

Invoke inefficient implementation of matrix multiplication
Elapsed time:   1.2630 sec (0.0010 sec resolution)
Profile completed ...
Generated raw file : C:\Temp\cpu-prof.prd

C:\Users\amd> AMDuProfCLI.exe report -i C:\Temp\cpu-prof.prd
Translation started ...
Translation done ...
Report generation started ...
Generating report file...

Report generation completed...

Generated report file : C:\Temp\cpu-prof\cpu-prof.csv

C:\Users\amd>
```

AMDuProfCLI – collect and report command invocations

This above screenshot shows how to run time-based profile and generate a report for the launch application AMDTClassicMatMul.exe.

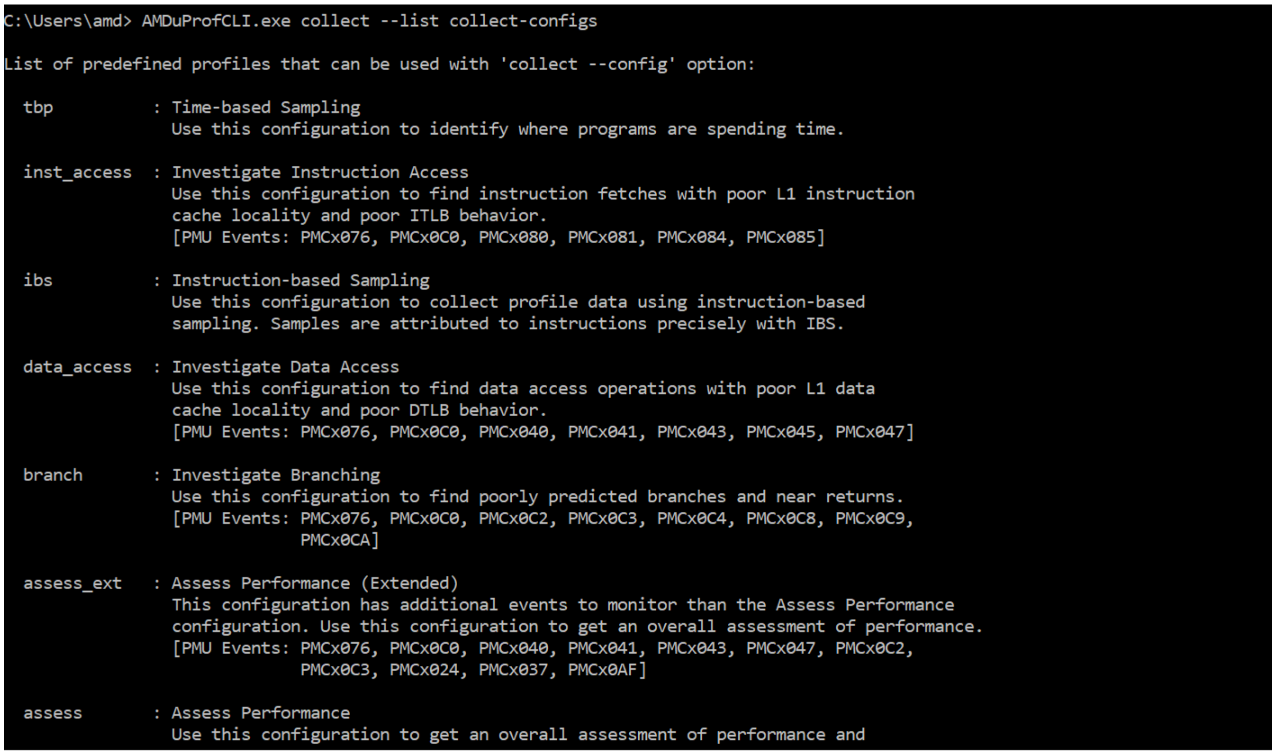
Note: On Linux, AMDuProfCLI **collect** command will generate **.caperf** file which will be passed as input file to **report** command.

List of predefined sampling configurations

To get the list of supported *predefined sampling configurations* that can be used with collect command's --config option run the below command.

```
C:\> AMDuProfCLI.exe collect -list collect-configs
```

And the output will look like:



```
C:\Users\amd> AMDuProfCLI.exe collect --list collect-configs

List of predefined profiles that can be used with 'collect --config' option:

  tbp          : Time-based Sampling
                 Use this configuration to identify where programs are spending time.

  inst_access  : Investigate Instruction Access
                 Use this configuration to find instruction fetches with poor L1 instruction
                 cache locality and poor ITLB behavior.
                 [PMU Events: PMCx076, PMCx0C0, PMCx080, PMCx081, PMCx084, PMCx085]

  ibs          : Instruction-based Sampling
                 Use this configuration to collect profile data using instruction-based
                 sampling. Samples are attributed to instructions precisely with IBS.

  data_access  : Investigate Data Access
                 Use this configuration to find data access operations with poor L1 data
                 cache locality and poor DTLB behavior.
                 [PMU Events: PMCx076, PMCx0C0, PMCx040, PMCx041, PMCx043, PMCx045, PMCx047]

  branch       : Investigate Branching
                 Use this configuration to find poorly predicted branches and near returns.
                 [PMU Events: PMCx076, PMCx0C0, PMCx0C2, PMCx0C3, PMCx0C4, PMCx0C8, PMCx0C9,
                              PMCx0CA]

  assess_ext   : Assess Performance (Extended)
                 This configuration has additional events to monitor than the Assess Performance
                 configuration. Use this configuration to get an overall assessment of performance.
                 [PMU Events: PMCx076, PMCx0C0, PMCx040, PMCx041, PMCx043, PMCx047, PMCx0C2,
                              PMCx0C3, PMCx024, PMCx037, PMCx0AF]

  assess       : Assess Performance
                 Use this configuration to get an overall assessment of performance and
```

AMDuProfCLI - list supported predefined configurations

Profile report

The profile report, which is CSV format, contains the following section:

- EXECUTION – information about the target lunch application
- PROFILE DETAILS – details about this session - profile type, scope, sampling events, etc.,
- 5 HOTTEST Functions – List of top 5 hot functions and the metrics attributed to them
- PROFILE REPORT FOR PROCESS – For the profiled process, the metrics attributed. This section contains other sub-sections like:
  - THREAD SUMMARY – list of threads that belongs to this process with metrics attributed to them
  - MODULE SUMMARY – list of load modules that belongs to this process with metrics attributed to them
  - FUNCTION SUMMARY – list of functions that belongs to this process for which samples are collected, with metrics attributed to them
  - Function Detail Data – Source level attribution for the top functions for which samples are collected
  - CALLGRAPH – Call graph, if callstack samples are collected

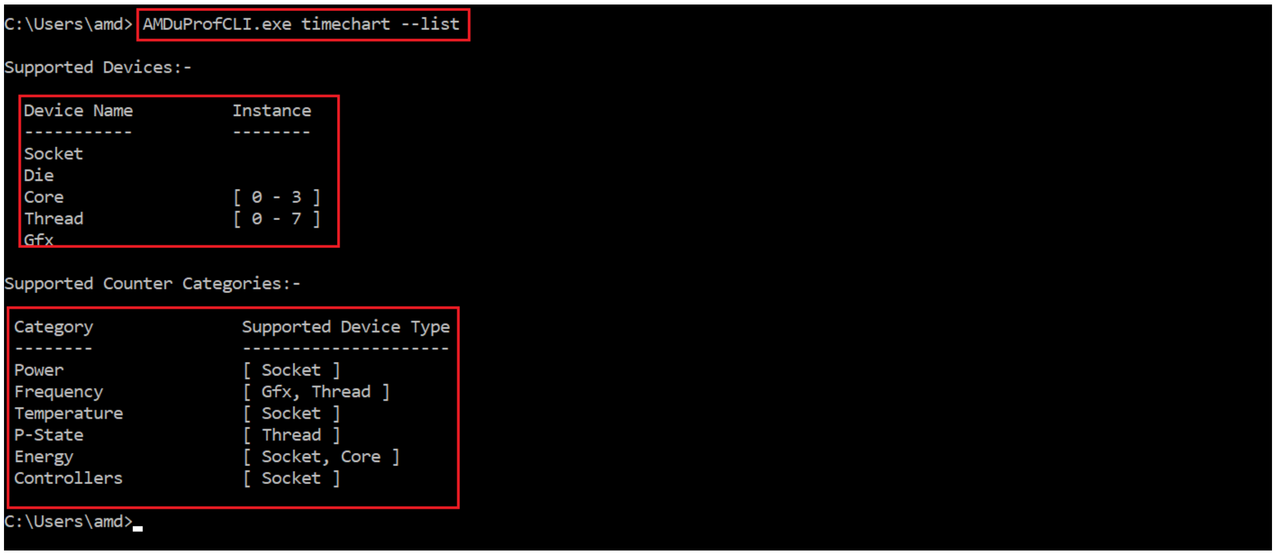
4.2 How to start Power profile?

System-wide Power Profiling (Live)

To collect power profile counter values, you need to follow these steps:

1. Get the list of supported counter categories by running AMDuProfCLI's **timechart** command with **--list** option
2. Collect and the report the required counters using AMDuProfCLI's timechart command by specifying the interesting counters with **--event** option

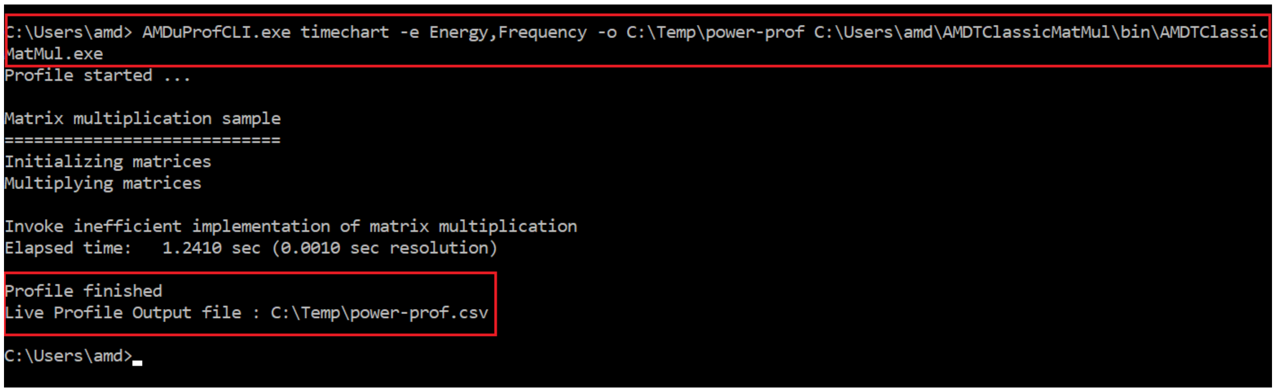
The timechart run to list the supported counter categories:



```
C:\Users\amd> AMDuProfCLI.exe timechart --list

Supported Devices:-

  Device Name          Instance
  -----------          --------
  Socket
  Die
  Core                 [ 0 - 3 ]
  Thread               [ 0 - 7 ]
  Gfx

Supported Counter Categories:-

 Category               Supported Device Type
 --------               ---------------------
 Power                  [ Socket ]
 Frequency              [ Gfx, Thread ]
 Temperature            [ Socket ]
 P-State                [ Thread ]
 Energy                 [ Socket, Core ]
 Controllers            [ Socket ]

C:\Users\amd>
```

AMDuProfCLI timechart --list command's output

The timechart to collect the profile samples and write into a file:



```
C:\Users\amd> AMDuProfCLI.exe timechart -e Energy,Frequency -o C:\Temp\power-prof C:\Users\amd\AMDTClassicMatMul\bin\AMDTClassic
MatMul.exe
Profile started ...

Matrix multiplication sample
============================
Initializing matrices
Multiplying matrices

Invoke inefficient implementation of matrix multiplication
Elapsed time:   1.2410 sec (0.0010 sec resolution)

Profile finished
Live Profile Output file : C:\Temp\power-prof.csv

C:\Users\amd>
```

AMDuProfCLI timechart run

The above run will collect the energy and frequency counters on all the devices on which these counters are supported and writes them in the output file specified with -o option. Before the profiling begins, the given application will be launched, and the data will be collected till the application terminates.

4.3 Collect command

This collect command runs the given program and collects the performance profile data and writes into specified raw profile data file. This file can then be analyzed using AMDuProfCLI's **report** command or AMDuProf GUI.

Synopsis:

```
AMDuProfCLI collect [--help] [--list <type>] [<options>] [<PROGRAM>]
[<ARGS>]
```

`<PROGRAM>` - Denotes a launch application to be profiled

`<ARGS>` - Denotes the list of arguments for the launch application

Common usages:

```
AMDuProfCLI collect --list <collect-configs | pmu-events>

AMDuProfCLI collect <PROGRAM> [<ARGS>]

AMDuProfCLI collect [--config <config> | -e <event>] [-a] [-d <duration>]
[<PROGRAM>]
```

Options:

Option	Description
`-h | --help`	Displays this help information on the console/terminal.
`--list <type>`	Lists the supported items for the following types: **collect-configs**: Predefined profile configurations that can be used with --config option. **pmu-events**: Raw PMU events that can be used with --event option.
`--config <config>`	Predefined sampling configuration to be used to collect samples. Use the command **collect --list collect-configs** to get the list of supported configs.

`-e | --event <EVENT>`	Specify a sampling event to monitor in the form of the comma separated key=value pair. Supported keys are: **event**=<timer | ibs-fetch | ibs-op | pmcxNNN> where NNN is hexadecimal PMC event id. **umask**=<unit-mask> **user**=<0 | 1> **os**=<0 | 1> **interval**=<sampling interval> **ibsop-count-control**=<0 | 1> **slicemask**=<L3 slice mask> **threadmask**=<L3 thread mask> Ex: **-e event=pmcx76,interval=250000** 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. Default is 0. **user, os** - Applicable to PMU events. Default is 1; **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. Default is 0. **ibsop-count-control** - Applicable only to ibs-op event. When set to 0, count clock cycles, otherwise count dispatched micro ops. Default is 0. **slicemask** - Applicable only to L3 PMU events. Default is 0xF. **threadmask** - Applicable only to L3 PMU events. Default is 0xFF. Multiple occurrences of --event (-e) are allowed.

	NOTE: L3 PMU events are supported only on Windows and only on Family 0x17 processors.
`-p | --pid <PID...>`	Profile existing processes (processes to attach to). Process IDs are separated by comma.
`-a | --system-wide`	System Wide Profile (SWP). If this flag is not set, then the command line tool will profile only the launched application, or the Process IDs attached with -p option.
`-c | --cpu <core...>`	Comma separated list of CPUs to profile. Ranges of CPUs also be specified with '-', e.g. 0-3. Use info --cpu-topology command to get list of available core-ids. *NOTE: On Windows, the selected cores should belong to only one processor group, e.g. 0-63, 64-127 and so on.*
`--call-graph <I:D:S:F>`	**[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.
`--call-graph <F:N>`	**[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 multiple of 8, then it is aligned down to the nearest value multiple of 8. The default value is 1024 bytes.

	NOTE: Passing a large N value will generate a very large raw data file. When **F** = **nofpo**, the value for **N** is ignored, hence no need to pass it.
`-g`	**[Windows]** Same as passing **--call-graph 1:128:user:nofpo** **[Linux]** Same as passing **--call-graph nofpo**
`-d | --duration <n>`	Profile only for the specified duration n in seconds.
`--affinity <core...>`	Set the core affinity of the launched application to be profiled. Comma separated list of core-ids. Ranges of core-ids also be specified, e.g. 0-3. Default affinity is all the available cores.
`--no-inherit`	Do not profile the children of the launched application (i.e. processes launched by the profiled application).
`-b | --terminate`	Terminate the launched application after profile data collection ends. Only the launched application process will be killed. Its children, if any, may continue to execute.
`--start-delay <n>`	Start Delay **n** in seconds. Start profiling after the specified duration. When n is 0, it has no impact.
`--start-paused`	Profiling paused indefinitely. The target application resumes the profiling using the profile control APIs. This option is expected to be used only when the launched application is instrumented to control the profile data collection using the resume and pause APIs defined in AMDProfileControl library.
`-w | --working-dir <path>`	Specify the working directory. Default will be the directory of the launch application.
`-o | --output <file>`	Base name of the output file. If this option is skipped, default path will be used. The default file will be (**Windows**) `%Temp%\AMDuProf-<timestamp>.prd` (**Linux**) `/tmp/AMDuProf-<timestamp>.caperf`
`-v | --verbose <n>`	Specify debug log messaging level. Valid values of (**n**) are: 1: INFO, 2: DEBUG, 3: EXTENSIVE

Examples

Windows:

- 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=0xF,threadmask=0xFF -a -d 10 -o
c:\Temp\cpuprof-l3
```

Linux:

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

  ```
  $ ./AMDuProfCLI collect -o /tmp/cpuprof-tbp AMDTClassicMatMul-bin
  ```

- Launch AMDTClassicMatMul-bin and do 'Assess Performance' profile for 10 seconds:

  ```
  $ ./AMDuProfCLI collect --config assess -o /tmp/cpuprof-assess -d 10
  AMDTClassicMatMul-bin
  ```

- Launch AMDTClassicMatMul-bin and collect 'IBS' samples in SWP mode:

  ```
  $ ./AMDuProfCLI collect --config ibs -a -o /tmp/cpuprof-ibs-swp
  AMDTClassicMatMul-bin
  ```

- Collect 'TBP' samples in SWP mode for 10 seconds:

  ```
  $ ./AMDuProfCLI collect -a -o /tmp/cpuprof-tbp-swp -d 10
  ```

- Launch AMDTClassicMatMul-bin and collect 'TBP' with Callstack sampling:

  ```
  $ ./AMDuProfCLI collect --config tbp -g -o /tmp/cpuprof-tbp
  AMDTClassicMatMul-bin
  ```

- Launch AMDTClassicMatMul-bin and collect 'TBP' with callstack sampling (unwind FPO optimized stack):

  ```
  $ ./AMDuProfCLI collect --config tbp --call-graph fpo:512 -o /tmp/uprof-
  tbp AMDTClassicMatMul-bin
  ```

- Launch AMDTClassicMatMul-bin and collect samples for PMCx076 and PMCx0C0:

  ```
  $ ./AMDuProfCLI collect -e event=pmcx76,interval=250000 -e
  event=pmcxc0,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
  ```

4.4 Report command

This report command processes the raw profile data (.prd on Windows or .caperf on Linux) or the processed file (.db) and generate a profile report. The profile report can also be generated from the DB file also.

Synopsis:

```
AMDuProfCLI report [--help] [--list view-configs] [<options>]
```

Common usages:

```
AMDuProfCLI report --list view-configs

AMDuProfCLI report -i <profile data file>
```

Options

Option	Description
`-h | --help`	Displays this help information on the console/terminal.
`--list view-configs`	List of the supported report view configurations that can be used with --view option.
`-i | --input <file>`	Input file name. Either the raw profile data file (**.prd** on Windows and **.caperf** on Linux) or the processed data file (**.db**) can be specified.
`-o | --output <output dir>`	Output directory in which the processed data file (**.db**) and the report file (**.csv**) will be created. The default output dir <base-name-of-input-file>, will be created in the directory in which the input file resides.
`--summary`	Report only the overview of the profile. This is set by default.
`--group-by <section>`	Specify the report to be generated. Supported report options are: **process**: Report process details **module**: Report module details **thread**: Report thread details Default is set to group-by **process**.
`--cutoff <n>`	Cutoff to limit the number of process, threads, modules and functions to be reported. n is the minimum number of entries to be reported in various report sections. Default value is 10.
`--view <config>`	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.
`--src`	Generate detailed function report with source statements.
`--src-path <path1;...>`	Source file directories. (Semicolon separated paths.)

`--disasm`	Generate detailed function report with assembly instructions.
`--sort-by <event-index>`	Specify the (0-based) event index on which the reported profile data will be sorted. This event is also used to generate Callgraph section and IMIX section. By default, the first event (i.e. event index 0) is selected.
`--imix`	Generate Instruction MIX report.
`--ignore-system-module`	Ignore samples from system modules.
`--show-percentage`	Show percentage of samples, instead of actual samples.
`--symbol-path <path1;...>`	Debug Symbol paths. (Semicolon separated paths.)
`--symbol-server <path1;...>`	**[Windows only]** Symbol Server directories. (Semicolon separated paths.)
`--symbol-cache-dir <path>`	**[Windows only]** Path to store the symbol files downloaded from the Symbol Servers.
`-v | --verbose <n>`	Specify debug log messaging level. Valid values are: 1 : INFO 2 : DEBUG 3 : EXTENSIVE

Examples

Windows

- Generate report from the raw datafile:

```
C:\> AMDuProfCLI.exe report -i c:\Temp\cpuprof-tbp.prd -o c:\Temp\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:\Temp\Symbols –symbol-
server http://msdl.microsoft.com/download/symbols --cache-dir C:\symbols -
i c:\Temp\cpuprof-tbp.prd -o c:\Temp\cpuprof-tbp-out
```

Linux

- 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
  ```

4.5 Timechart command

This **timechart** command collects and reports system characteristics like power, thermal and frequency metrics and generates a text or CSV report.

Synopsis:

```
AMDuProfCLI timechart [--help] [--list] [<options>] [<PROGRAM>] [<ARGS>]
```

`<PROGRAM>` - Denotes the application to be launch before start collecting the power metrics

`<ARGS>` - Denotes the list of arguments for the launch application

Common usages:

```
AMDuProfCLI timechart --list

AMDuProfCLI timechart -e <event> -d <duration> [<PROGRAM>] [<ARGS>]
```

Options:

Option	Description
`--list`	Display all the supported devices and categories.
`-e | --event <type...>`	Collect counters for specified type or comma separated list of types, where type can be a device or a category. **Supported device list:** **socket**: Collect profile data from socket. **die**: Collect profile data from die. **core**: Collect profile data from core. **thread**: Collect profile data from thread.

	Supported category list: Refer *this* section for family specific supported categories. **power**: Collect all available power counters. **frequency**: Collect all available frequency counters. **temperature**: Collect all available temperature counters. **voltage**: Collect all available voltage counters. **current**: Collect all available current counters. **dvfs**: Collect all available Dynamic Voltage and Frequency Scaling (DVFS) counters. **energy**: Collect all available energy counters. **correlatedpower**: Collect all available correlated power counters. **cac**: Collect all available cac counters. **controllers**: Collect all available controllers counters. Note: Multiple occurrences of -e is allowed.
`--histogram`	Collect histogram counters. Allowed only with 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...>`	Core affinity. Comma separated list of core-ids. Ranges of core-ids also be specified, e.g. 0-3. Default affinity is all the available cores. Affinity is set for the launched application.
`-w | --working-dir <dir>`	Set the working directory for the launched target application.
`-f | --format <fmt>`	Output file format. Supported formats are:

	txt: Text (.txt) format. csv: Comma Separated Value (.csv) format. Default file format is CSV.
`-o | --output <file>`	Output file path.
`-h | --help`	Displays this help information.

Examples:

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\output.txt --
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
```

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
```

4.6 Info command

This **info** command helps to get generic information about the system, CPU topology, disassembly of a binary etc.

Synopsis:

```
AMDuProfCLI info [--help] [<options>]
```

Common usages:

```
AMDuProfCLI info --system

AMDuProfCLI info --cpu-topology
```

Options:

Option	Description
`--help`	Displays the help information.
`--system`	Displays processor information of this system.
`--cpu-topology`	Displays CPU topology information of this system.
`--disasm <binary>`	Disassembles the given binary file.
`--show-uid`	Displays the UID of the user.

Examples:

- Print system details:
  ```
  C:\> AMDuProfCLI.exe info --system
  ```
- Print CPU topology details:
  ```
  C:\> AMDuProfCLI.exe info --cpu-topology
  ```
- To disassemble AMDTClassicMatMul.exe into classic-disasm.txt file:
  ```
  C:\> AMDuProfCLI.exe info --disasm AMDTClassicMatMul.exe > classic_asm.txt
  ```

Chapter 5 Performance Analysis

CPU Profiling

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) - precise instruction-based sampling
- Call-stack Sampling
- Secondary profile data (Windows only)
    - Thread concurrency
    - Thread Names
- Profile scope
    - 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
- Profiles
    - C, C++, Java, .NET, FORTRAN, Assembly applications
    - Various software components – Applications, Dynamically linked/loaded modules, Driver, OS Kernel modules
- 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.

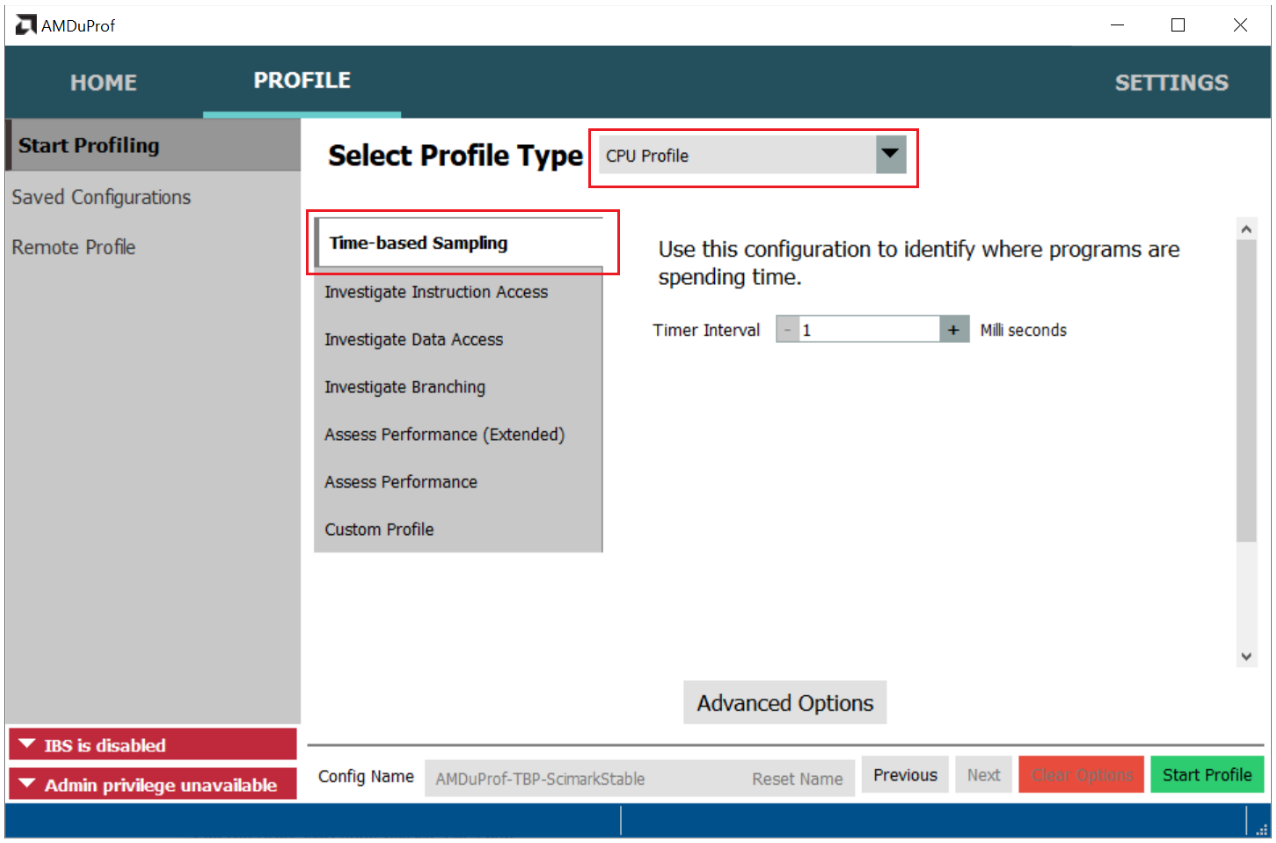
- **AMDuProfCLI**, the command-line-interface can be used to configure a profile run, collect the profile data and generate the profile report.
    - **collect** option to configure and collect the profile data
    - **report** option to process the profile data and to generate the profile report
- **AMDuProf** 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
    - Hot spots summary
    - Thread concurrency graph (Windows only and requires admin privileges)
    - Process and function analysis
    - Source and disassembly 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

5.1 Analysis with Time-based profiling

In this analysis, the profile data is periodically collected based on the specified OS timer interval. It is used to identify the hotspots of the profiled applications that are consuming the most time. These hotspots are good candidates for further investigation and optimization. Follow these steps:

To configure and start profile:

1. 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. In **Start Profiling** window, you will see **Select Profile Target** fragment. After selecting the appropriate profile target, clicking **Next** button will take you to **Select Profile Type** fragment.
3. In **Select Profile Type** fragment, selecting **CPU Profile** from the drop-down list, will take you to the below screenshot.
4. Select **Time-based Sampling** in the left vertical pane as shown in the below screenshot.



Use this configuration to identify where programs are spending time.

Timer Interval 1 Milli seconds

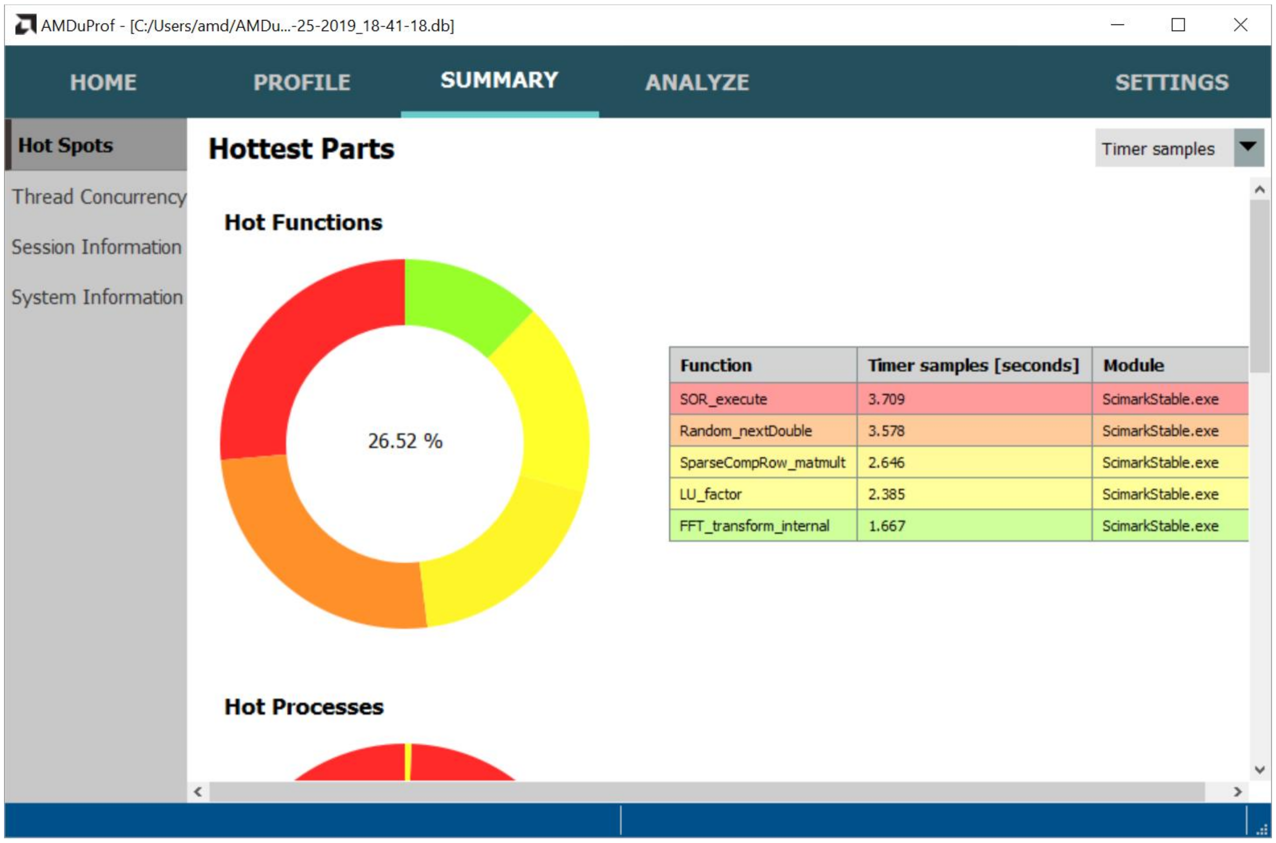
Config Name AMDuProf-TBP-ScimarkStable Reset Name

Time based profile – configure

5. Once all the options are set, the **Start Profile** button at the bottom will be enabled and you can click on it to start the profile. After the profile initialization you will see *this* profile data collection screen.

To Analyze the profile data

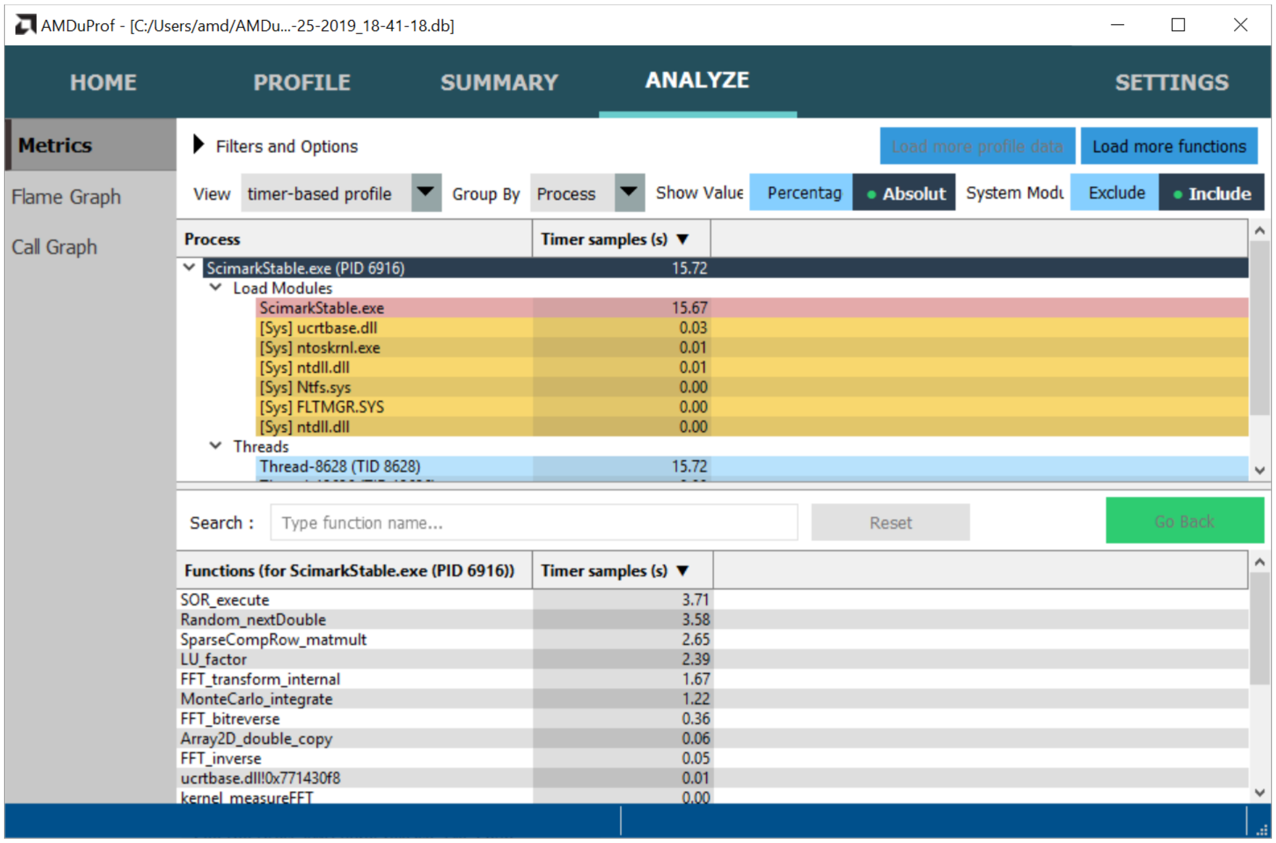
6. When the profiling stopped, the collected raw profile data will be processed automatically, and you will the **Hot spots** window of **Summary** page. The hotspots are shown for **Timer** samples. Refer *this* section for more information on this window.



Function	Timer samples [seconds]	Module
SOR_execute	3.709	ScimarkStable.exe
Random_nextDouble	3.578	ScimarkStable.exe
SparseCompRow_matmult	2.646	ScimarkStable.exe
LU_factor	2.385	ScimarkStable.exe
FFT_transform_internal	1.667	ScimarkStable.exe

SUMMARY – Hot Spots window of TBP profile

7. Click on the **ANALYZE** button on the top horizontal navigation bar to go **Metrics** window, which displays the profile data table at various granularities - Process, Load Modules, Threads and Functions. Refer *this* section for more information on this window.



AMDuProf - [C:/Users/amd/AMDu...-25-2019_18-41-18.db]

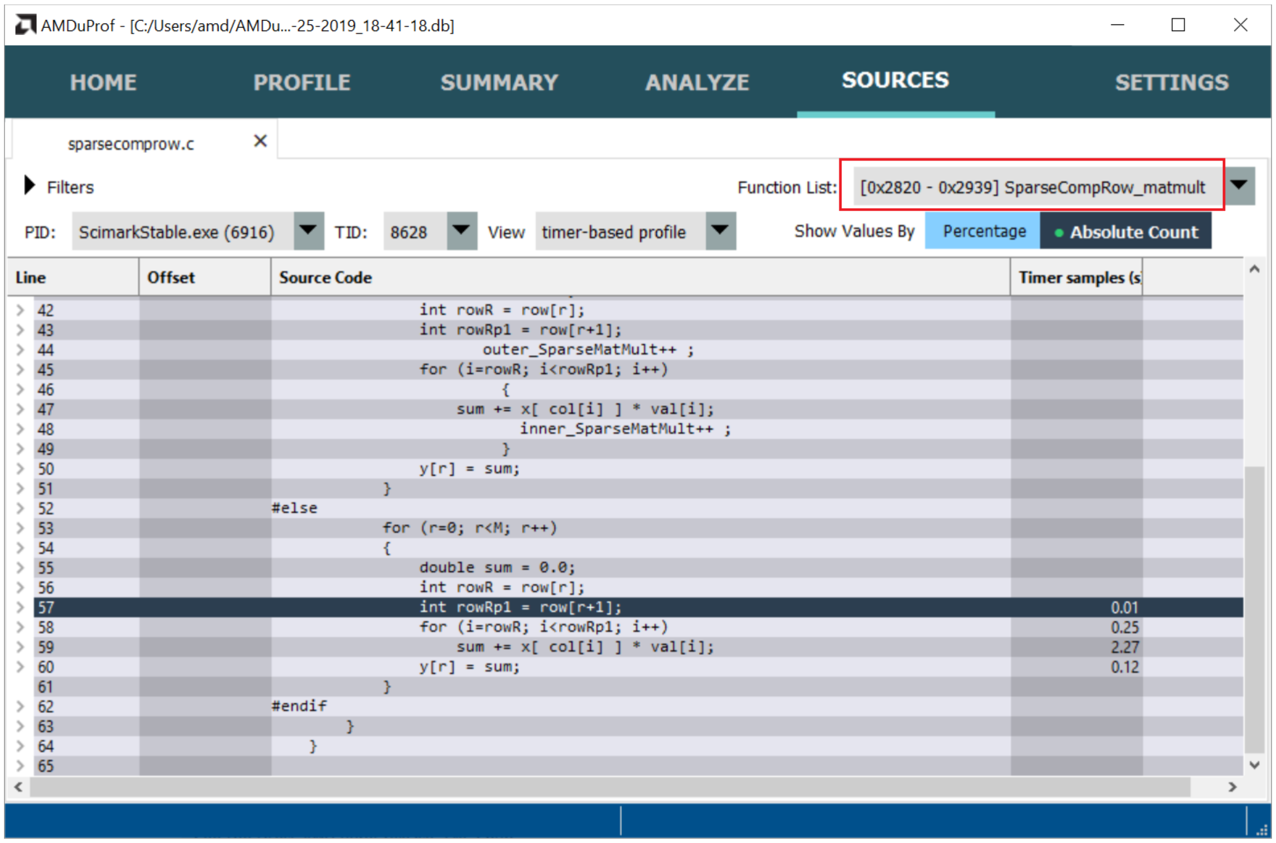
View: timer-based profile | Group By: Process | Show Value: Percentag / ● Absolut | System Modu: Exclude / ● Include

Process	Timer samples (s) ▼
ScimarkStable.exe (PID 6916)	15.72
Load Modules	
ScimarkStable.exe	15.67
[Sys] ucrtbase.dll	0.03
[Sys] ntoskrnl.exe	0.01
[Sys] ntdll.dll	0.01
[Sys] Ntfs.sys	0.00
[Sys] FLTMGR.SYS	0.00
[Sys] ntdll.dll	0.00
Threads	
Thread-8628 (TID 8628)	15.72

Functions (for ScimarkStable.exe (PID 6916))	Timer samples (s) ▼
SOR_execute	3.71
Random_nextDouble	3.58
SparseCompRow_matmult	2.65
LU_factor	2.39
FFT_transform_internal	1.67
MonteCarlo_integrate	1.22
FFT_bitreverse	0.36
Array2D_double_copy	0.06
FFT_inverse	0.05
ucrtbase.dll!0x771430f8	0.01
kernel_measureFFT	0.00

ANALYZE page - Metrics window

8. Double-clicking any entry on the **Functions** table in **Metrics** window will make the GUI load the source tab for that function in **SOURCES** page. Refer *this* section for more information on this window.



Function List: [0x2820 - 0x2939] SparseCompRow_matmult

PID: ScimarkStable.exe (6916) TID: 8628 View timer-based profile Show Values By Percentage Absolute Count

```
Line  Offset  Source Code                                        Timer samples (s
42                    int rowR = row[r];
43                    int rowRp1 = row[r+1];
44                        outer_SparseMatMult++ ;
45                    for (i=rowR; i<rowRp1; i++)
46                         {
47                      sum += x[ col[i] ] * val[i];
48                            inner_SparseMatMult++ ;
49                         }
50                    y[r] = sum;
51                }
52      #else
53                for (r=0; r<M; r++)
54                {
55                    double sum = 0.0;
56                    int rowR = row[r];
57                    int rowRp1 = row[r+1];                              0.01
58                    for (i=rowR; i<rowRp1; i++)                         0.25
59                      sum += x[ col[i] ] * val[i];                      2.27
60                    y[r] = sum;                                         0.12
61                }
62      #endif
63            }
64          }
65
```

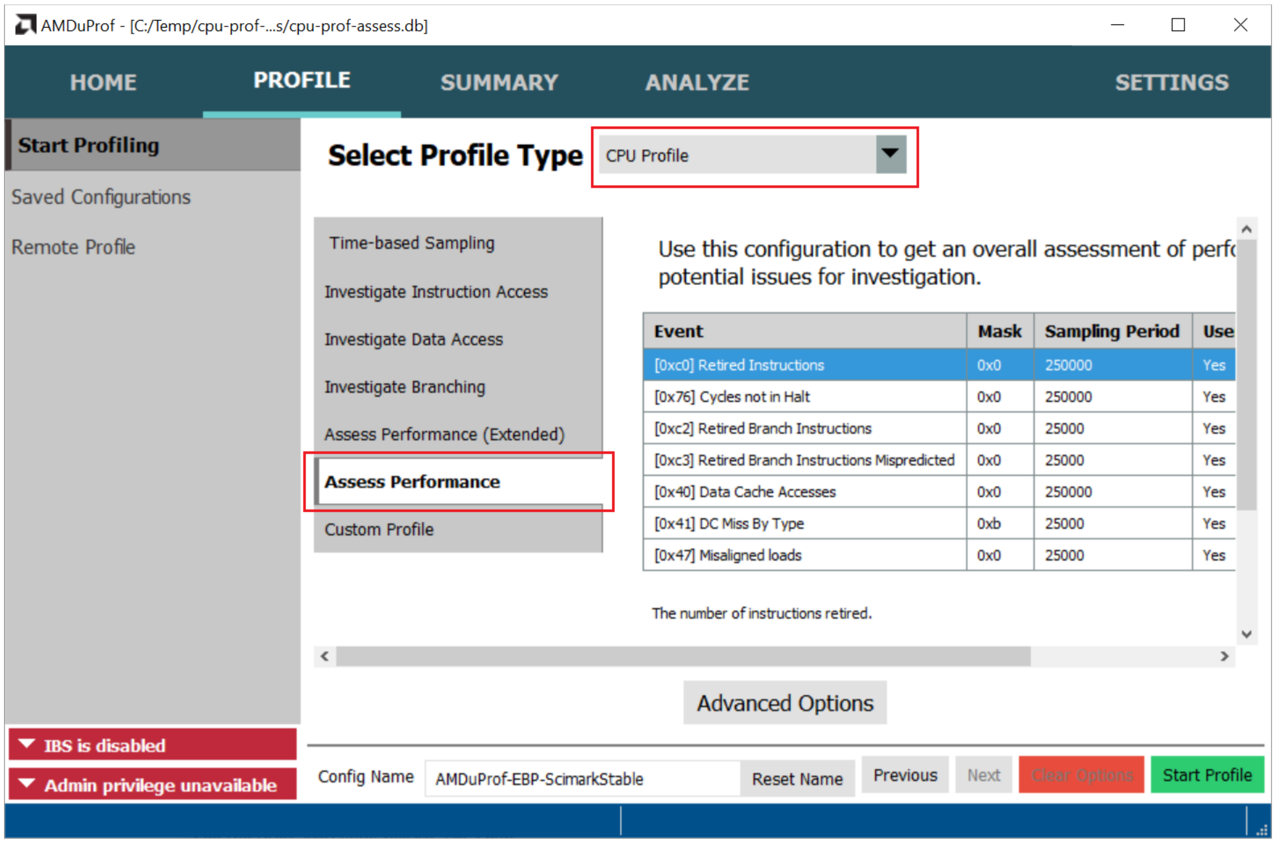
SOURCES – source and assembly window

5.2 Analysis with Event based profiling

In this profile, the uProf 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. Steps to follow:

To configure and start profile:

1. 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. In **Start Profiling** window, you will see **Select Profile Target** fragment. After selecting the appropriate profile target, clicking **Next** button will take you to **Select Profile Type** fragment.
3. In **Select Profile Type** fragment, selecting **CPU Profile** from the drop-down list, will take you to the below screenshot.
4. Select **Assess Performance** in the left vertical pane as shown in the below screenshot. Refer *this* section for EBP based predefined sampling configurations.



Use this configuration to get an overall assessment of perf
potential issues for investigation.

Event	Mask	Sampling Period	Use
[0xc0] Retired Instructions	0x0	250000	Yes
[0x76] Cycles not in Halt	0x0	250000	Yes
[0xc2] Retired Branch Instructions	0x0	25000	Yes
[0xc3] Retired Branch Instructions Mispredicted	0x0	25000	Yes
[0x40] Data Cache Accesses	0x0	250000	Yes
[0x41] DC Miss By Type	0xb	25000	Yes
[0x47] Misaligned loads	0x0	25000	Yes

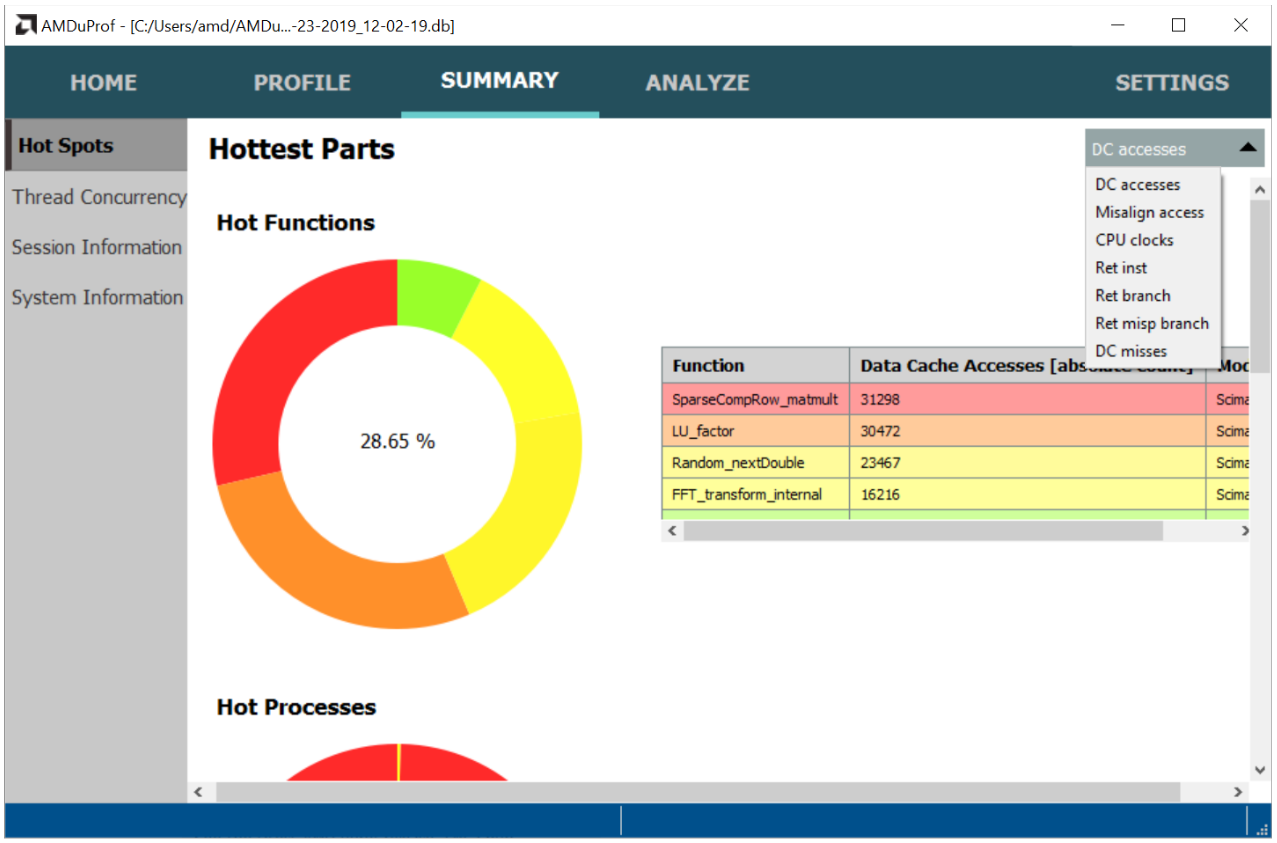
The number of instructions retired.

Event based profile - configure

5. Once all the options are set, the **Start Profile** button at the bottom will be enabled and you can click on it to start the profile. After the profile initialization you will see *this* profile data collection screen.

To Analyze the profile data

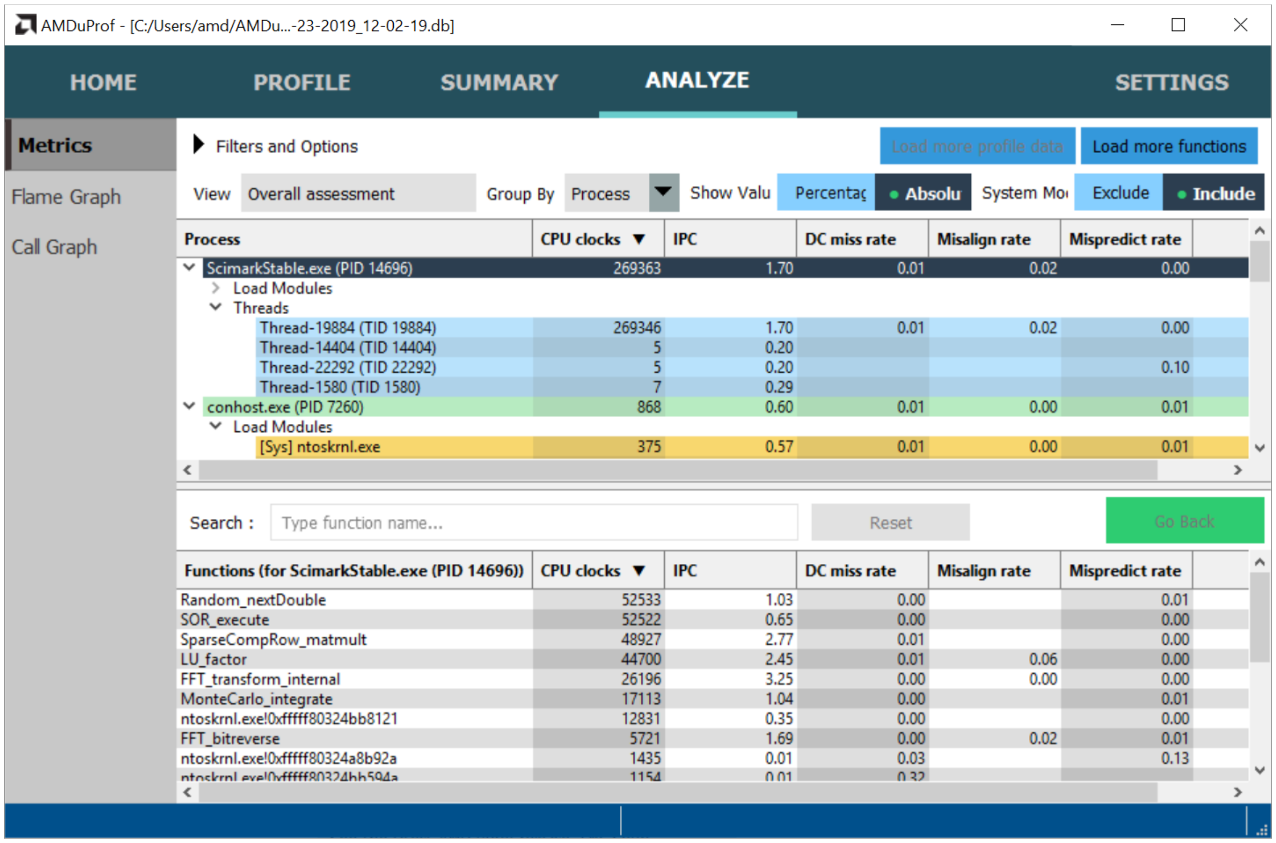
6. When the profiling stopped, the collected raw profile data will be processed automatically, and you will the **Hot spots** window of **Summary** page. Refer *this* section for more information on this window.



Function	Data Cache Accesses [abs	Moc
SparseCompRow_matmult	31298	Scima
LU_factor	30472	Scima
Random_nextDouble	23467	Scima
FFT_transform_internal	16216	Scima

SUMMARY – Hot Spots window

7. Click on the **ANALYZE** button on the top horizontal navigation bar to go **Metrics** window, which displays the profile data table at various granularities - Process, Load Modules, Threads and Functions. Refer *this* section for more information on this window.



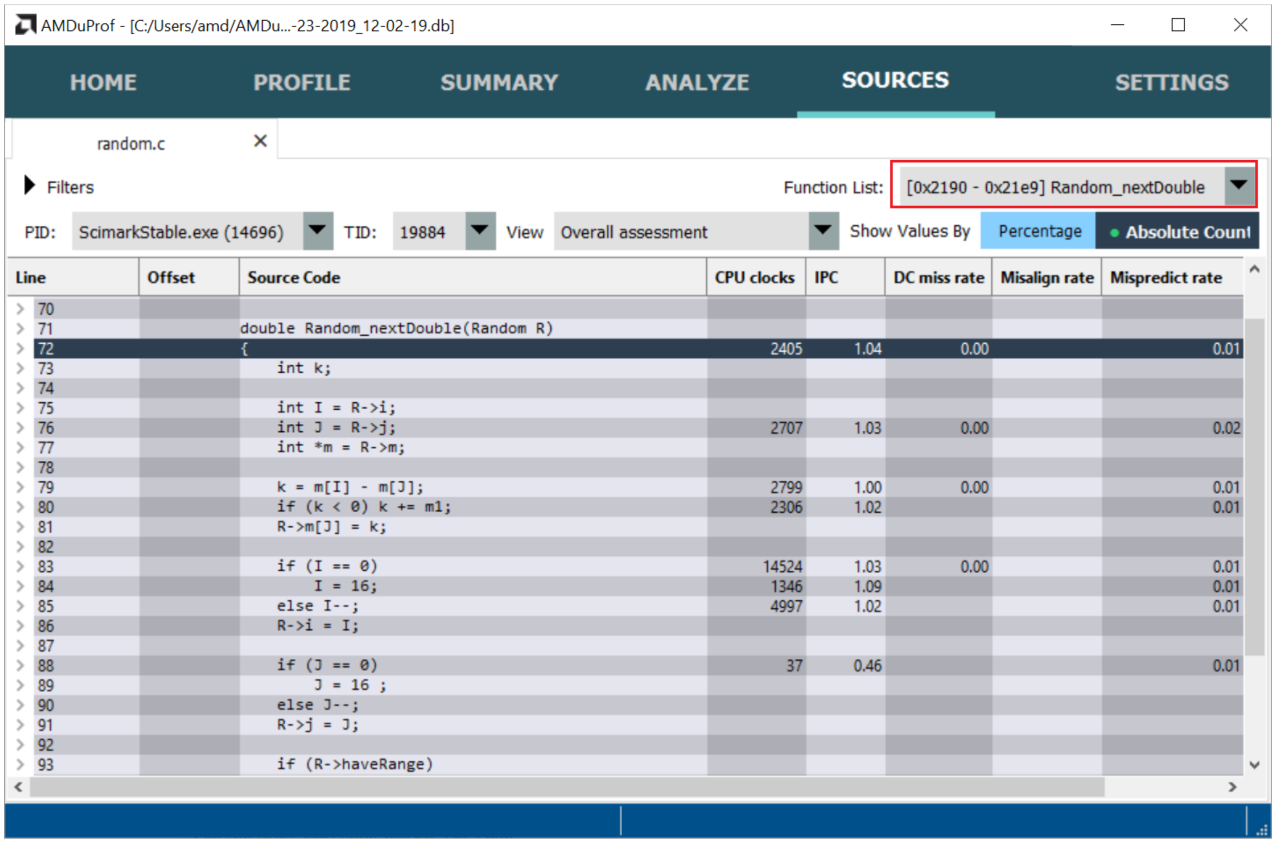
View: Overall assessment · Group By: Process · Show Valu: Percentag / ● Absolu · System Mo: Exclude / ● Include

Process	CPU clocks ▼	IPC	DC miss rate	Misalign rate	Mispredict rate
ScimarkStable.exe (PID 14696)	269363	1.70	0.01	0.02	0.00
Load Modules					
Threads					
Thread-19884 (TID 19884)	269346	1.70	0.01	0.02	0.00
Thread-14404 (TID 14404)	5	0.20			
Thread-22292 (TID 22292)	5	0.20			0.10
Thread-1580 (TID 1580)	7	0.29			
conhost.exe (PID 7260)	868	0.60	0.01	0.00	0.01
Load Modules					
[Sys] ntoskrnl.exe	375	0.57	0.01	0.00	0.01

Functions (for ScimarkStable.exe (PID 14696))	CPU clocks ▼	IPC	DC miss rate	Misalign rate	Mispredict rate
Random_nextDouble	52533	1.03	0.00		0.01
SOR_execute	52522	0.65	0.00		0.00
SparseCompRow_matmult	48927	2.77	0.01		0.00
LU_factor	44700	2.45	0.01	0.06	0.00
FFT_transform_internal	26196	3.25	0.00	0.00	0.00
MonteCarlo_integrate	17113	1.04	0.00		0.01
ntoskrnl.exe!0xfffff80324bb8121	12831	0.35	0.00		0.00
FFT_bitreverse	5721	1.69	0.00	0.02	0.01
ntoskrnl.exe!0xfffff80324a8b92a	1435	0.01	0.03		0.13
ntoskrnl.exe!0xfffff80324bb594a	1154	0.01	0.32		

ANALYZE page - Metrics window

8. Double-clicking any entry on the **Functions** table in **Metrics** window will make the GUI load the source tab for that function in **SOURCES** page. Refer *this* section for more information on this window.



Function List: [0x2190 - 0x21e9] Random_nextDouble

PID: ScimarkStable.exe (14696) TID: 19884 View Overall assessment Show Values By Percentage ● Absolute Count

Line	Offset	Source Code	CPU clocks	IPC	DC miss rate	Misalign rate	Mispredict rate
70							
71		`double Random_nextDouble(Random R)`					
72		`{`	2405	1.04	0.00		0.01
73		`int k;`					
74							
75		`int I = R->i;`					
76		`int J = R->j;`	2707	1.03	0.00		0.02
77		`int *m = R->m;`					
78							
79		`k = m[I] - m[J];`	2799	1.00	0.00		0.01
80		`if (k < 0) k += m1;`	2306	1.02			0.01
81		`R->m[J] = k;`					
82							
83		`if (I == 0)`	14524	1.03	0.00		0.01
84		`I = 16;`	1346	1.09			0.01
85		`else I--;`	4997	1.02			0.01
86		`R->i = I;`					
87							
88		`if (J == 0)`	37	0.46			0.01
89		`J = 16 ;`					
90		`else J--;`					
91		`R->j = J;`					
92							
93		`if (R->haveRange)`					

SOURCES – source and assembly window

5.3 Analysis with Instruction based sampling

In this profile, the uProf uses the IBS supported by the AMD x86-based processor to diagnose the performance issues in hot spots. It collects data on how instructions behave on the processor and in the memory subsystem.

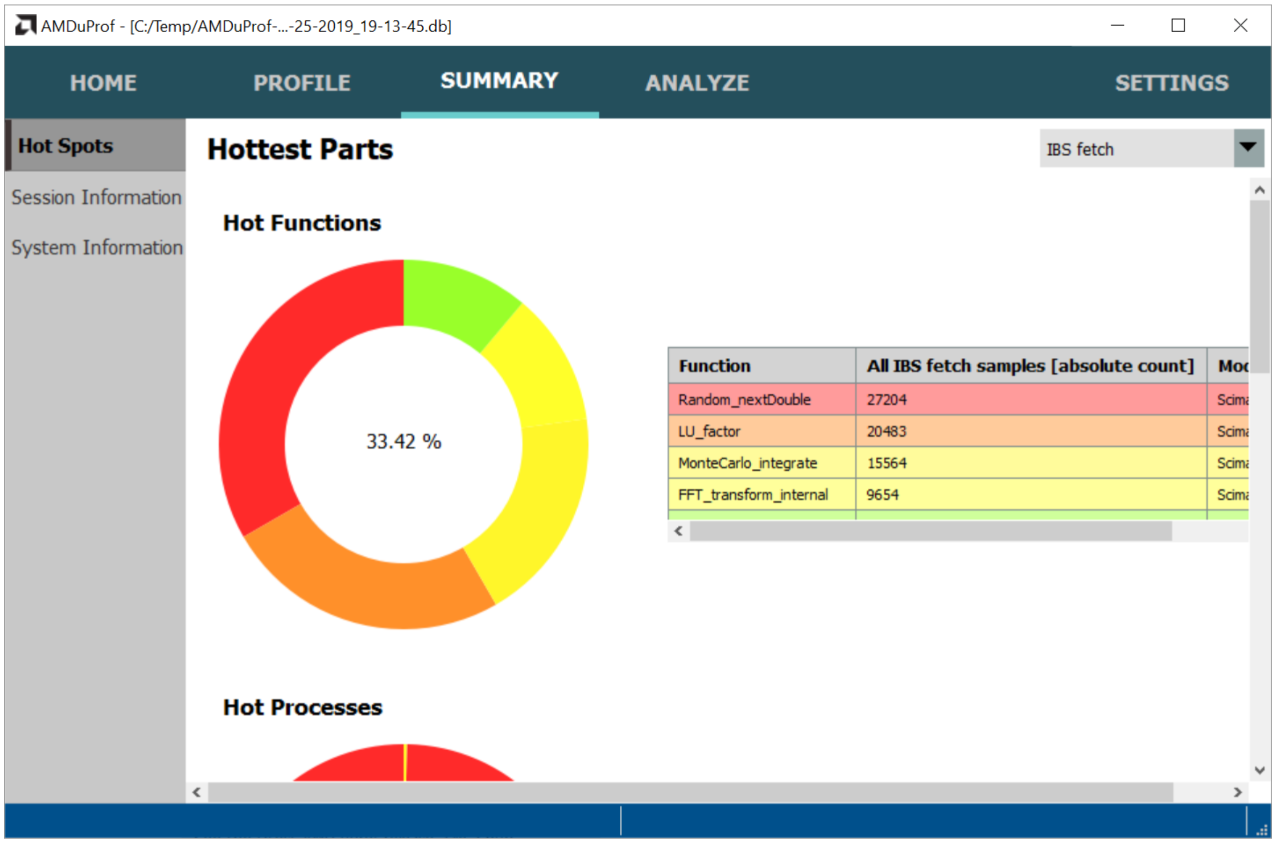
To configure and start profile:

1. 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. In **Start Profiling** window, you will see **Select Profile Target** fragment. After selecting the appropriate profile target, clicking **Next** button will take you to **Select Profile Type** fragment.
3. In **Select Profile Type** fragment, selecting **CPU Profile** from the drop-down list, will take you to the below screenshot.
4. Select **Instruction-Based Sampling** in the left vertical pane. Refer *this* section for predefined sampling configurations.

5. Once all the options are set, the **Start Profile** button at the bottom will be enabled and you can click on it to start the profile. After the profile initialization you will see *this* profile data collection screen.

To Analyze the profile data

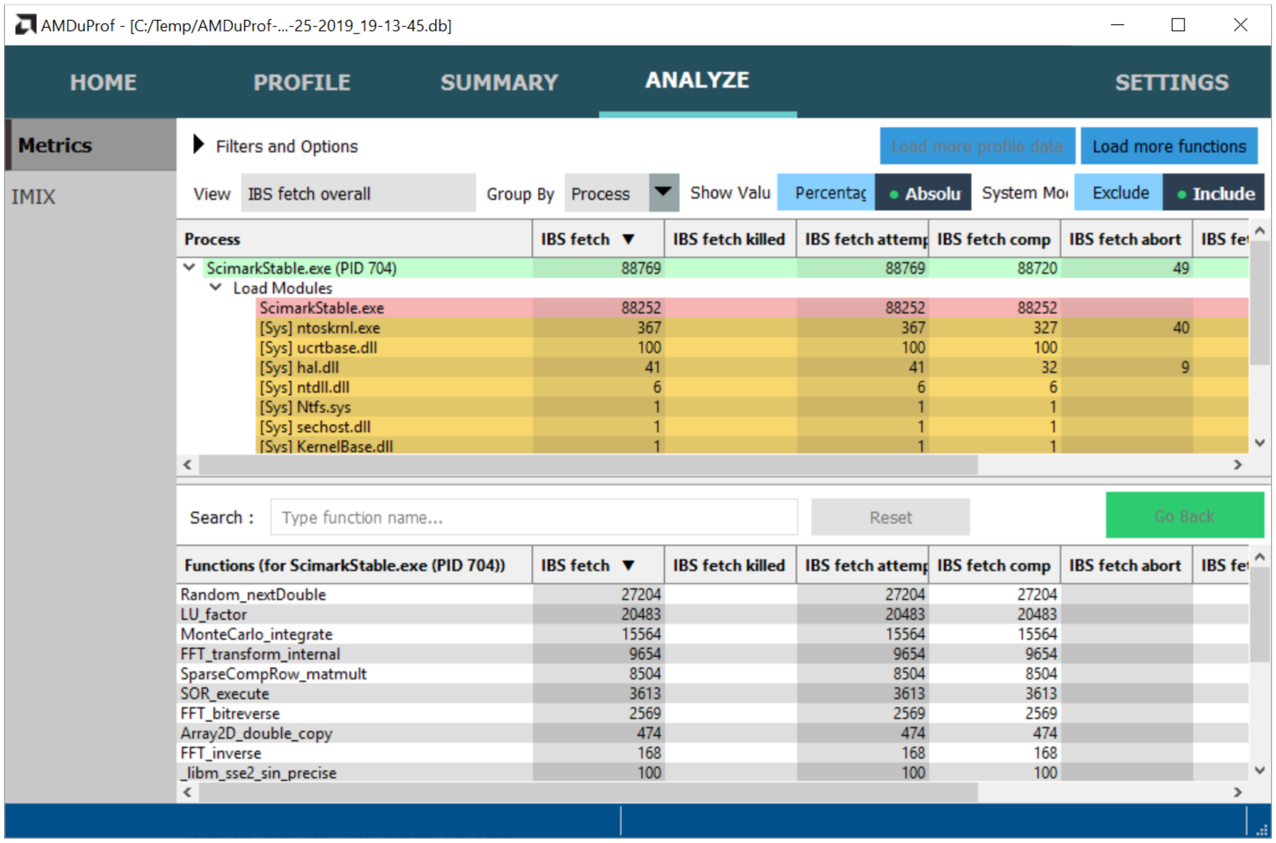
6. When the profiling stopped, the collected raw profile data will be processed automatically, and you will the **Hot spots** window of **Summary** page. Refer *this* section for more information on this window.



Function	All IBS fetch samples [absolute count]	Moc
Random_nextDouble	27204	Scima
LU_factor	20483	Scima
MonteCarlo_integrate	15564	Scima
FFT_transform_internal	9654	Scima

SUMMARY – Hot Spots window

7. Click on the **ANALYZE** button on the top horizontal navigation bar to go **Metrics** window, which displays the profile data table at various granularities - Process, Load Modules, Threads and Functions. Refer *this* section for more information on this window.


AMDuProf - [C:/Temp/AMDuProf-...-25-2019_19-13-45.db]

View: IBS fetch overall Group By: Process Show Valu: Percentag / ● Absolu System Mo: Exclude / ● Include

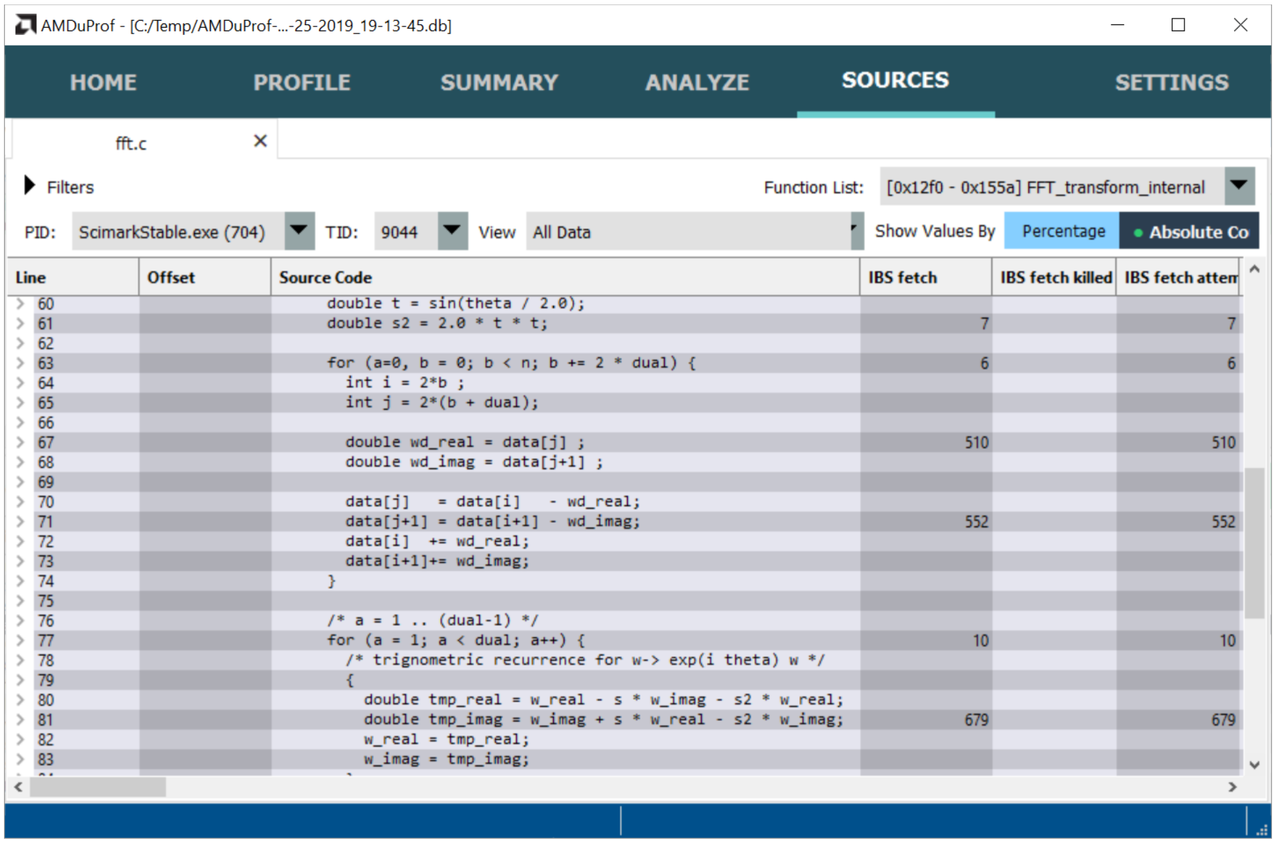
Process	IBS fetch ▼	IBS fetch killed	IBS fetch attemp	IBS fetch comp	IBS fetch abort	IBS fe
ScimarkStable.exe (PID 704)	88769		88769	88720	49	
Load Modules						
ScimarkStable.exe	88252		88252	88252		
[Sys] ntoskrnl.exe	367		367	327	40	
[Sys] ucrtbase.dll	100		100	100		
[Sys] hal.dll	41		41	32	9	
[Sys] ntdll.dll	6		6	6		
[Sys] Ntfs.sys	1		1	1		
[Sys] sechost.dll	1		1	1		
[Sys] KernelBase.dll	1		1	1		

Search : Type function name...

Functions (for ScimarkStable.exe (PID 704))	IBS fetch ▼	IBS fetch killed	IBS fetch attemp	IBS fetch comp	IBS fetch abort	IBS fe
Random_nextDouble	27204		27204	27204		
LU_factor	20483		20483	20483		
MonteCarlo_integrate	15564		15564	15564		
FFT_transform_internal	9654		9654	9654		
SparseCompRow_matmult	8504		8504	8504		
SOR_execute	3613		3613	3613		
FFT_bitreverse	2569		2569	2569		
Array2D_double_copy	474		474	474		
FFT_inverse	168		168	168		
_libm_sse2_sin_precise	100		100	100		

ANALYZE page - Metrics window for IBS profile run

8. Double-clicking any entry on the **Functions** table in **Metrics** window will make the GUI load the source tab for that function in **SOURCES** page. Refer *this* section for more information on this window.



Function List: [0x12f0 - 0x155a] FFT_transform_internal

PID: ScimarkStable.exe (704) TID: 9044 View All Data Show Values By

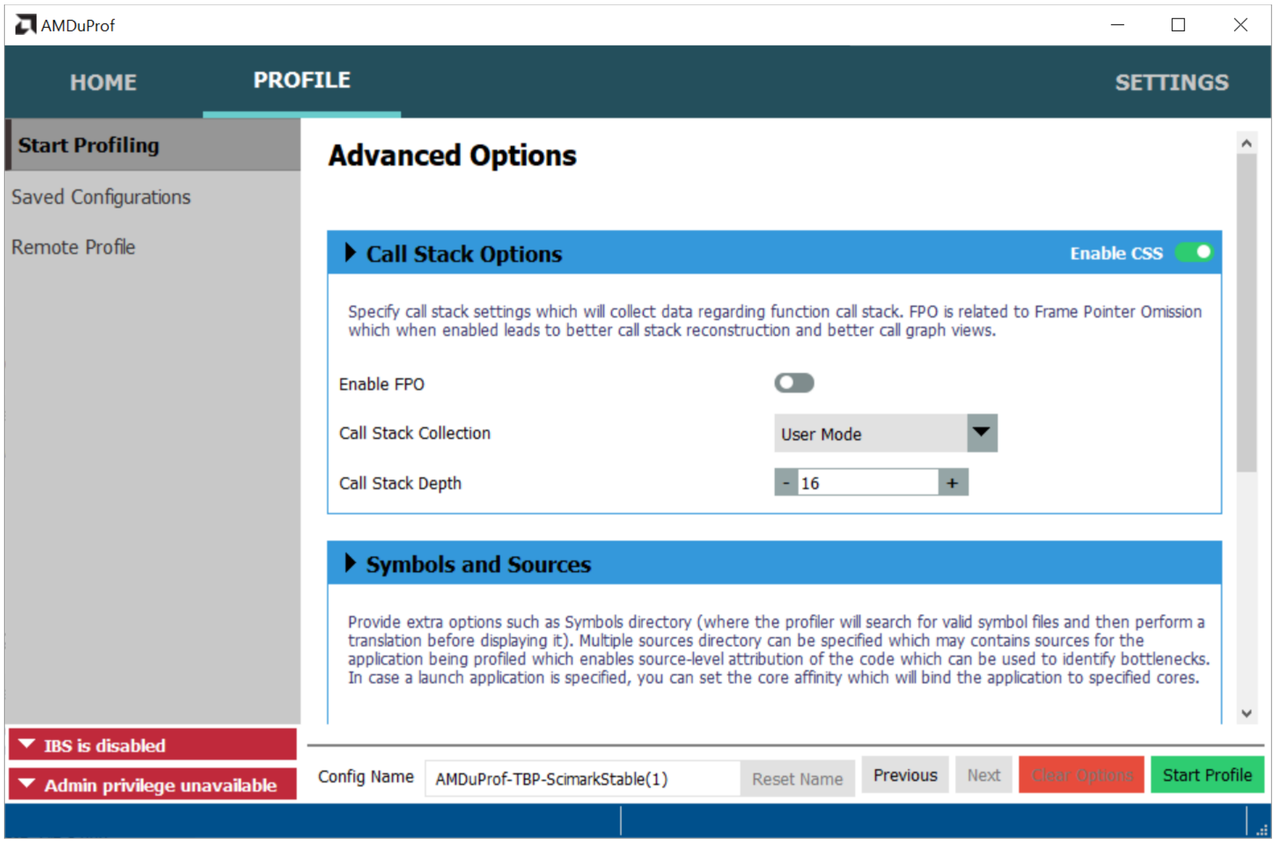
Line	Offset	Source Code	IBS fetch	IBS fetch killed	IBS fetch attem
60		`double t = sin(theta / 2.0);`			
61		`double s2 = 2.0 * t * t;`	7		7
62					
63		`for (a=0, b = 0; b < n; b += 2 * dual) {`	6		6
64		`int i = 2*b ;`			
65		`int j = 2*(b + dual);`			
66					
67		`double wd_real = data[j] ;`	510		510
68		`double wd_imag = data[j+1] ;`			
69					
70		`data[j]   = data[i]   - wd_real;`			
71		`data[j+1] = data[i+1] - wd_imag;`	552		552
72		`data[i]  += wd_real;`			
73		`data[i+1]+= wd_imag;`			
74		`}`			
75					
76		`/* a = 1 .. (dual-1) */`			
77		`for (a = 1; a < dual; a++) {`	10		10
78		`/* trignometric recurrence for w-> exp(i theta) w */`			
79		`{`			
80		`double tmp_real = w_real - s * w_imag - s2 * w_real;`			
81		`double tmp_imag = w_imag + s * w_real - s2 * w_imag;`	679		679
82		`w_real = tmp_real;`			
83		`w_imag = tmp_imag;`			

SOURCES – source and assembly window

5.4 Analysis with Callstack samples

The callstack samples too can be collected for native C & C++ applications with all the CPU profile types. These samples will be used to provide Flame Graph and Call Graph window.

To enable call-stack sampling, after selecting profile target and profile type, click on **Advanced Options** button to turn on the **Enable CSS** option in **Call Stack Options** pane, as seen in the below screen.



Advanced Options

Call Stack Options — Enable CSS

Specify call stack settings which will collect data regarding function call stack. FPO is related to Frame Pointer Omission which when enabled leads to better call stack reconstruction and better call graph views.

Enable FPO

Call Stack Collection: User Mode

Call Stack Depth: 16

Symbols and Sources

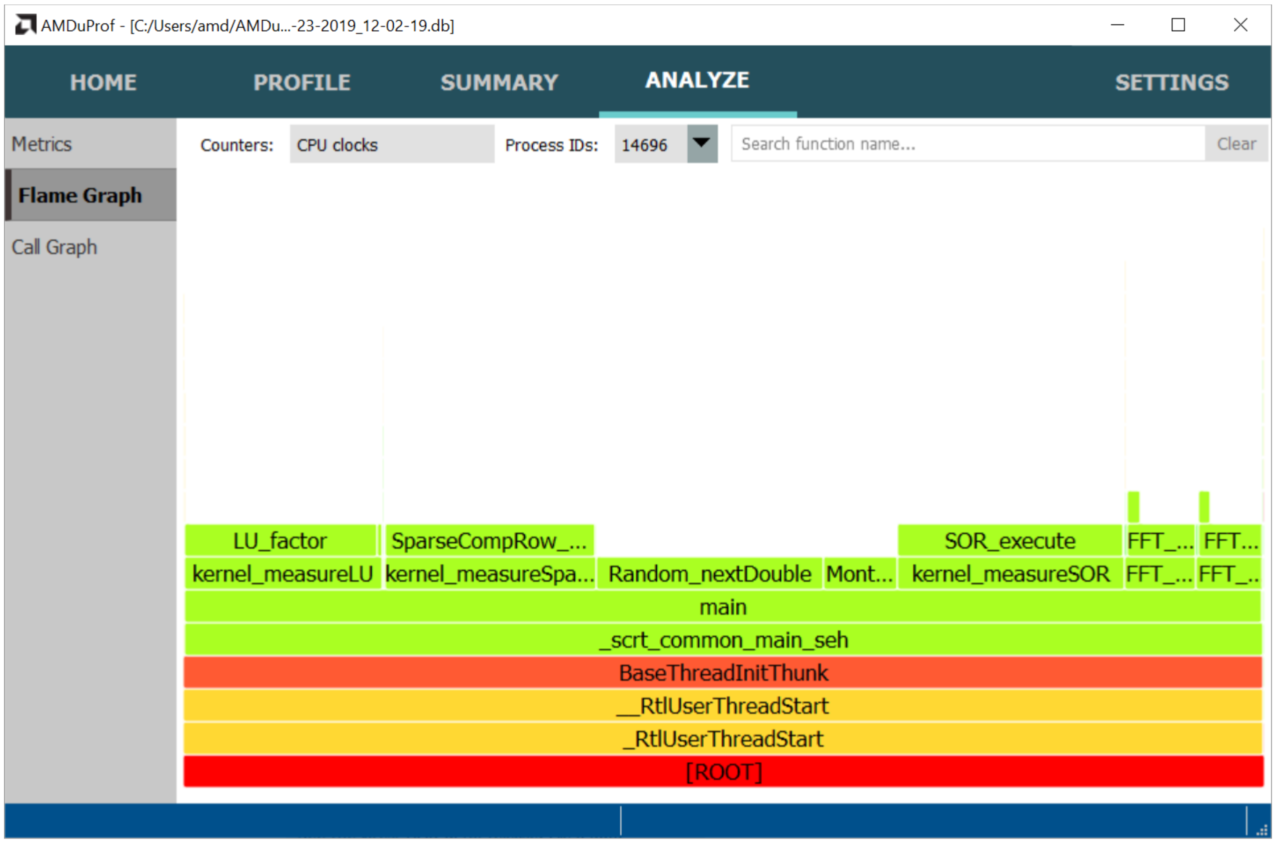
Provide extra options such as Symbols directory (where the profiler will search for valid symbol files and then perform a translation before displaying it). Multiple sources directory can be specified which may contains sources for the application being profiled which enables source-level attribution of the code which can be used to identify bottlenecks. In case a launch application is specified, you can set the core affinity which will bind the application to specified cores.

Config Name: AMDuProf-TBP-ScimarkStable(1)

Start Profiling – Advanced Options

5.4.1 Flame graph

Flame Graph provides a stack visualizer based on call-stack samples. The **Flame Graph** window will be available in **ANALYZE** page to analyze the call-stack samples to identify hot call-paths. It can be navigated by clicking **ANALYZE → Flame Graph** in the left vertical pane.

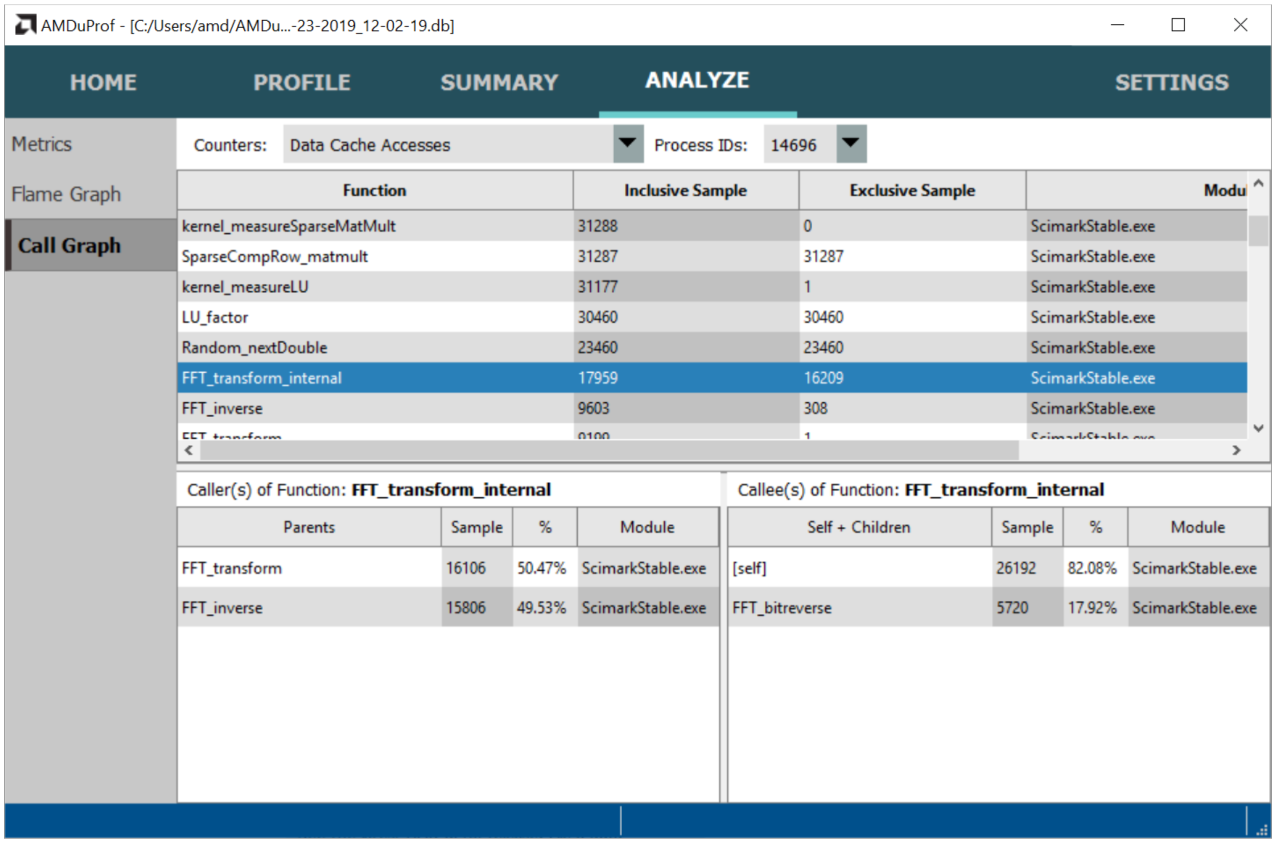


ANALYZE – Flame graph window

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.

5.4.2 Call graph

Call Graph provides a butterfly view of callgraph based on call-stack samples The **Call Graph** window will be available in **ANALYZE** page to analyze the call-stack samples to identify hot call-paths. It can be navigated by clicking **ANALYZE → Call Graph** in the left vertical pane.



AMDuProf - [C:/Users/amd/AMDu...-23-2019_12-02-19.db]

HOME PROFILE SUMMARY ANALYZE SETTINGS

Metrics | Flame Graph | Call Graph

Counters: Data Cache Accesses Process IDs: 14696

Function	Inclusive Sample	Exclusive Sample	Modu
kernel_measureSparseMatMult	31288	0	ScimarkStable.exe
SparseCompRow_matmult	31287	31287	ScimarkStable.exe
kernel_measureLU	31177	1	ScimarkStable.exe
LU_factor	30460	30460	ScimarkStable.exe
Random_nextDouble	23460	23460	ScimarkStable.exe
FFT_transform_internal	17959	16209	ScimarkStable.exe
FFT_inverse	9603	308	ScimarkStable.exe
FFT_transform	9100	1	ScimarkStable.exe

Caller(s) of Function: **FFT_transform_internal**

Parents	Sample	%	Module
FFT_transform	16106	50.47%	ScimarkStable.exe
FFT_inverse	15806	49.53%	ScimarkStable.exe

Callee(s) of Function: **FFT_transform_internal**

Self + Children	Sample	%	Module
[self]	26192	82.08%	ScimarkStable.exe
FFT_bitreverse	5720	17.92%	ScimarkStable.exe

ANALYZE – Call graph window

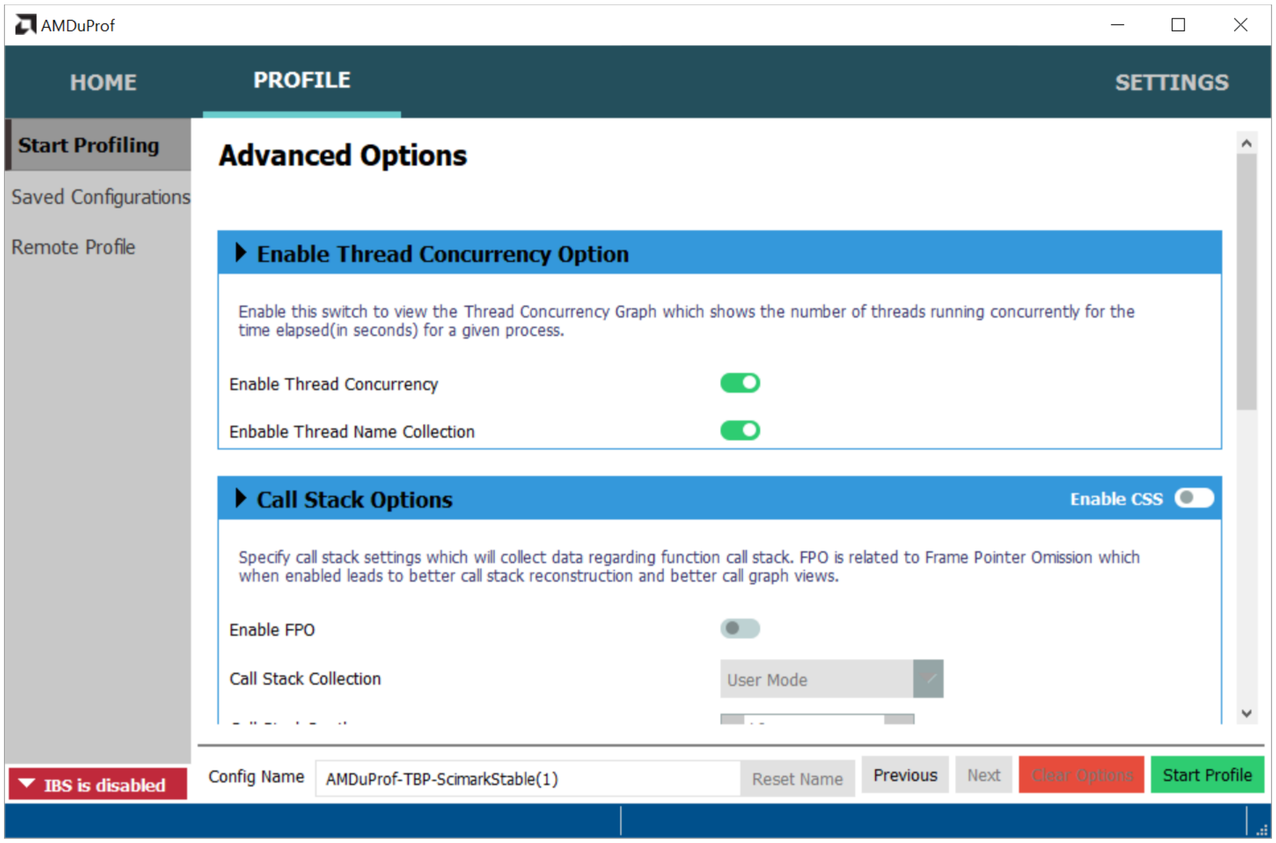
The data can be browsed based on **Process IDs** and **Counters** drop-downs. The top central table displays call-stack samples for each function. Clicking on any function updates the bottom two **Caller(s)** and **Callee(s)** tables. These tables display the callers and callees respectively of the selected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

5.5 Thread Concurrency

Thread concurrency graph shows the number of threads of a process, running concurrently for the time elapsed (in seconds). It uses Windows ETL records to generate this graph. It is:

- A Windows OS only feature that requires **Admin privileges**
- Available only with **CPU Profile** types

To enable this, after selecting profile target and profile type, click on **Advanced Options** button to turn on the **Enable Thread Concurrency** option in **Enable Thread Concurrency Option** pane, as seen in the below screen.



Advanced Options

Enable Thread Concurrency Option

Enable this switch to view the Thread Concurrency Graph which shows the number of threads running concurrently for the time elapsed(in seconds) for a given process.

Enable Thread Concurrency

Enbable Thread Name Collection

Call Stack Options

Enable CSS

Specify call stack settings which will collect data regarding function call stack. FPO is related to Frame Pointer Omission which when enabled leads to better call stack reconstruction and better call graph views.

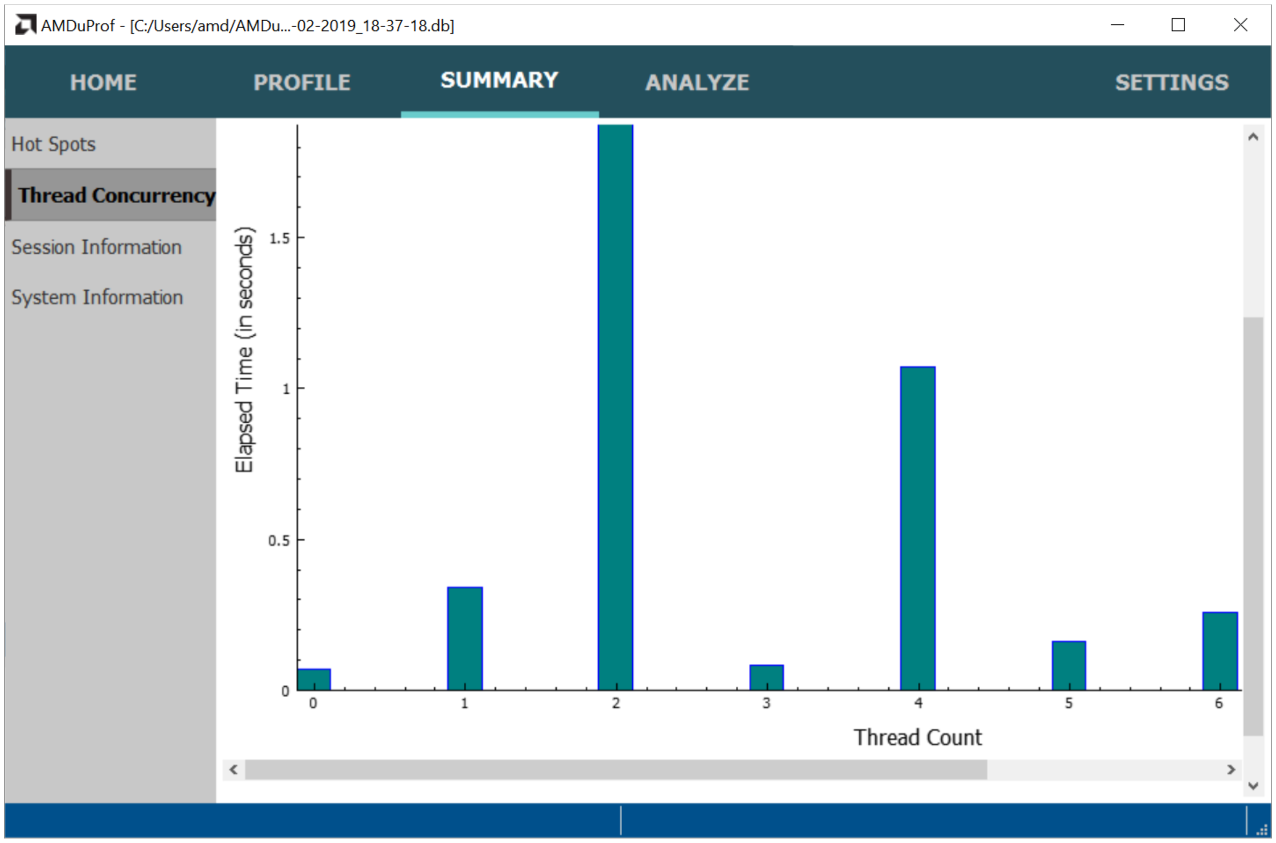
Enable FPO

Call Stack Collection: User Mode

Config Name: AMDuProf-TBP-ScimarkStable(1)

Start Profiling – Advanced Options

After the profile completion, clicking **SUMMARY → Thread Concurrency** will take you to the following window to analyze the thread concurrency of the application.



SUMMARY – Thread Concurrency

5.6 Profiling a Java Application

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

AMDuProf provides JVMTI Agent libraries: `AMDJvmtiAgent.dll` on Windows and `libAMDJvmtiAgent.so` on Linux. This JvmtiAgent library needs to be loaded during start-up of the target JVM process.

Launching a Java application

If the Java application is launched by uProf, then the tool would take care of passing the AMDJvmtiAgent library to JVM using Java's -agentpath option. AMDuProf would be able to collect the profile data and attribute the samples to interpreted Java functions.

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>
```

Attaching a Java process to profile

AMD uProf can't attach JvmtiAgent dynamically to an already running JVM. Hence any JVM process profiled by attach-process mechanism, uProf can't capture any class information, unless the JvmtiAgent library is loaded during JVM process start-up.

If you want to profile an already running Java process, then you must pass -agentpath <path to agent lib> option while launching Java application. So that, later uProf can attach to the Java PID to collect profile data.

For a 64-bit JVM on Linux:

```
$ java
-agentpath:<AMDuProf-install-dir/bin/ProfileAgents/x64/libAMDJvmtiAgent.so>
<java-app-launch-options>
```

For a 64-bit JVM on Windows:

```
C:\> java -agentpath:
<C:\ProgramFiles\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 AMDuProf GUI or AMDuProfCLI to attach to this process and profile.

5.7 Profiling Linux System Modules

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.

- Ubuntu: *https://wiki.ubuntu.com/Debug%20Symbol%20Packages*
- SLES/OpenSUSE: *https://www.suse.com/support/kb/doc/?id=3074997*
- RHEL/CentOS: *https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Developer_Guide/intro.debuginfo.html*

Make sure to download the debug info files for the required system modules for the required Linux distros before starting the profiling.

5.8 Profiling Linux Kernel

To attribute the kernel samples to appropriate kernel functions, uProf extracts 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.

To attribute the kernel samples to kernel functions, before profiling, you need to:

- Set kptr_restrict file to 0, or
- Set kptr_restrict file to 1, if the current user has a CAP_SYSLOG capability

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"

5.9 Limitations

- 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).

Chapter 6 System Analysis

System-wide Power Profile

AMD uProf profiler offers live power profiling to monitor the behavior of the systems based on AMD CPUs, APUs and dGPUs. It provides various counters to monitor power and thermal characteristics.

These counters are collected from various resources like RAPL, SMU and MSRs. These are periodically collected at regular timer interval and either reported as text file or plotted as line graphs and can also be saved into DB for future analysis.

Features

- AMDuProf GUI can be used to configure and monitor the supported 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 – RAPL based Core Energy, Temperature
  - Package level metrics – RAPL based Package Energy
  - GPU metrics – power, temperature, frequency
  - SMU based APU metrics – CPU Core power, package power
- 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 CPU/APUs and dGPUs.
- Collected live profile data can be stored in database for future analysis

6.1 Metrics

The metrics that are supported depends on the processor family and model and they are broadly grouped under various categories. Following are supported counter categories for various processor families:

Family 17h Model 00h – 0Fh (Ryzen, ThreadRipper, EPIC 7001)

Power Counter Category	Description
Power	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
Frequency	Core Effective Frequency for the sampling period, reported in MHz
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 Die
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 and Power
CorrelatedPower	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

Family 17h Model 10h – 2Fh (Ryzen APU, Ryzen PRO APU)

Power Counter Category	Description
Power	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
Frequency	Core Effective Frequency for the sampling period, reported in MHz
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

Family 17h Model 70h – 7Fh (3rd Gen Ryzen)

Power Counter Category	Description
Frequency	Core Effective Frequency for the sampling period, reported in MHz
P-State	CPU Core P-State at the time when sampling was performed
Energy	RAPL MSRs based Package and Core energy

Family 17h Model 30h – 3Fh (EPIC 7002)

Power Counter Category	Description
Frequency	Core Effective Frequency for the sampling period, reported in MHz
P-State	CPU Core P-State at the time when sampling was performed
Energy	RAPL MSRs based Package and Core energy

Supported Counter categories for older APU families

Power Counter Category	Description
Power	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
Frequency	Effective Frequency for the sampling period, reported in MHz Available for Core and iGPU
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 CPU ComputeUnit and iGPU
P-State	CPU Core P-State at the time when sampling was performed
Controllers	Socket PPT Limit and Power

CorrelatedPower	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

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

6.2 Profile using GUI

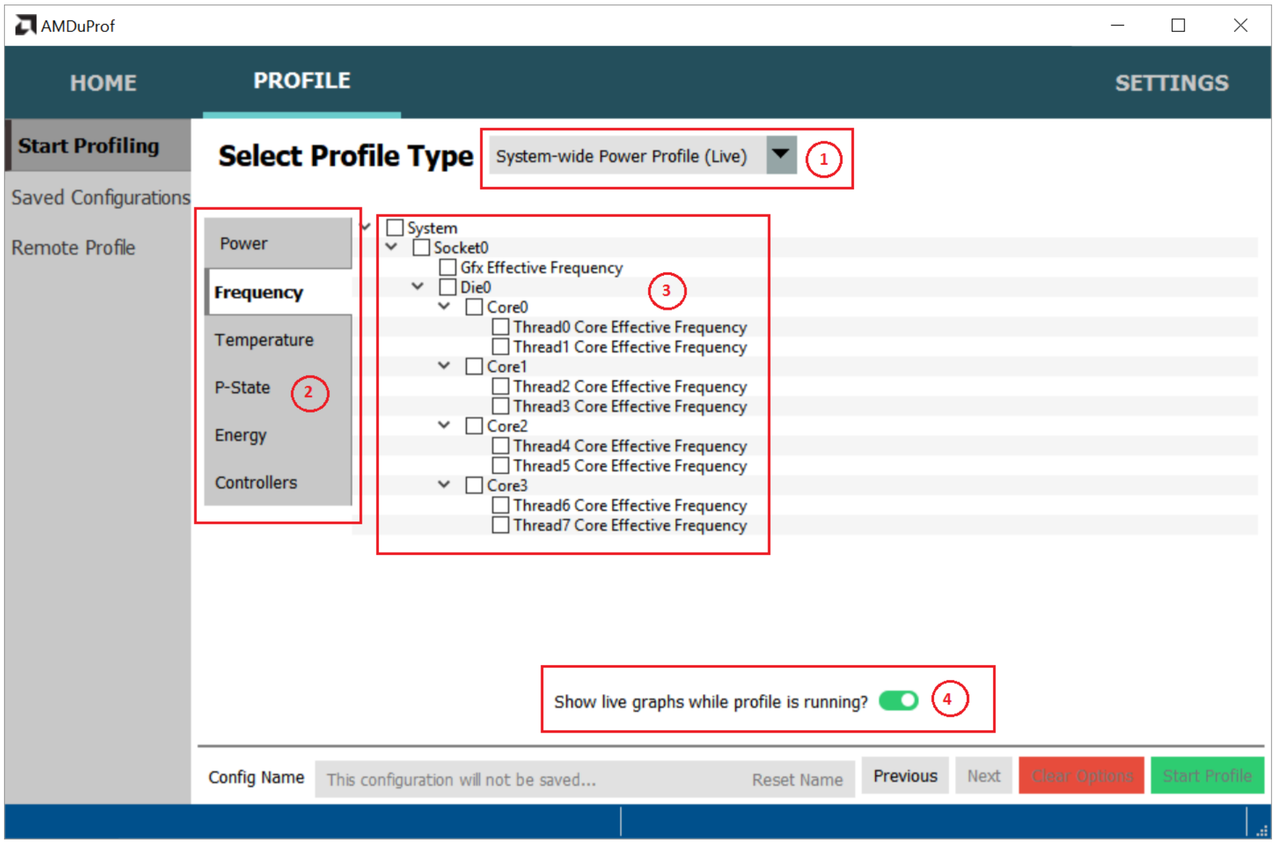
System-wide Power Profile (Live): This profile type is used to perform the system analysis where the metrics are plotted in a live timeline graph and/or saved in a DB. Here are the steps to configure and start the profile:

6.2.1 Configure

- 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.
- You will see **Select Profile Target** fragment in the **Start Profiling** window. After selecting the appropriate profile target, clicking **Next** button will take you to **Select Profile Type** fragment.
- In **Select Profile Type** fragment selecting **System-wide Power Profile (Live)** from the drop-down list, will take you to the below screenshot.

You can also navigate to this page by clicking **See what's guzzling power in your System** link in the **Welcome** page.

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 to monitor.


AMDuProf
HOME
PROFILE
SETTINGS
Start Profiling
Saved Configurations
Remote Profile
Select Profile Type
System-wide Power Profile (Live)
1
Power
Frequency
Temperature
P-State
2
Energy
Controllers
System
Socket0
Gfx Effective Frequency
Die0
3
Core0
Thread0 Core Effective Frequency
Thread1 Core Effective Frequency
Core1
Thread2 Core Effective Frequency
Thread3 Core Effective Frequency
Core2
Thread4 Core Effective Frequency
Thread5 Core Effective Frequency
Core3
Thread6 Core Effective Frequency
Thread7 Core Effective Frequency
Show live graphs while profile is running?
4
Config Name
This configuration will not be saved...
Reset Name
Previous
Next
Clear Options
Start Profile

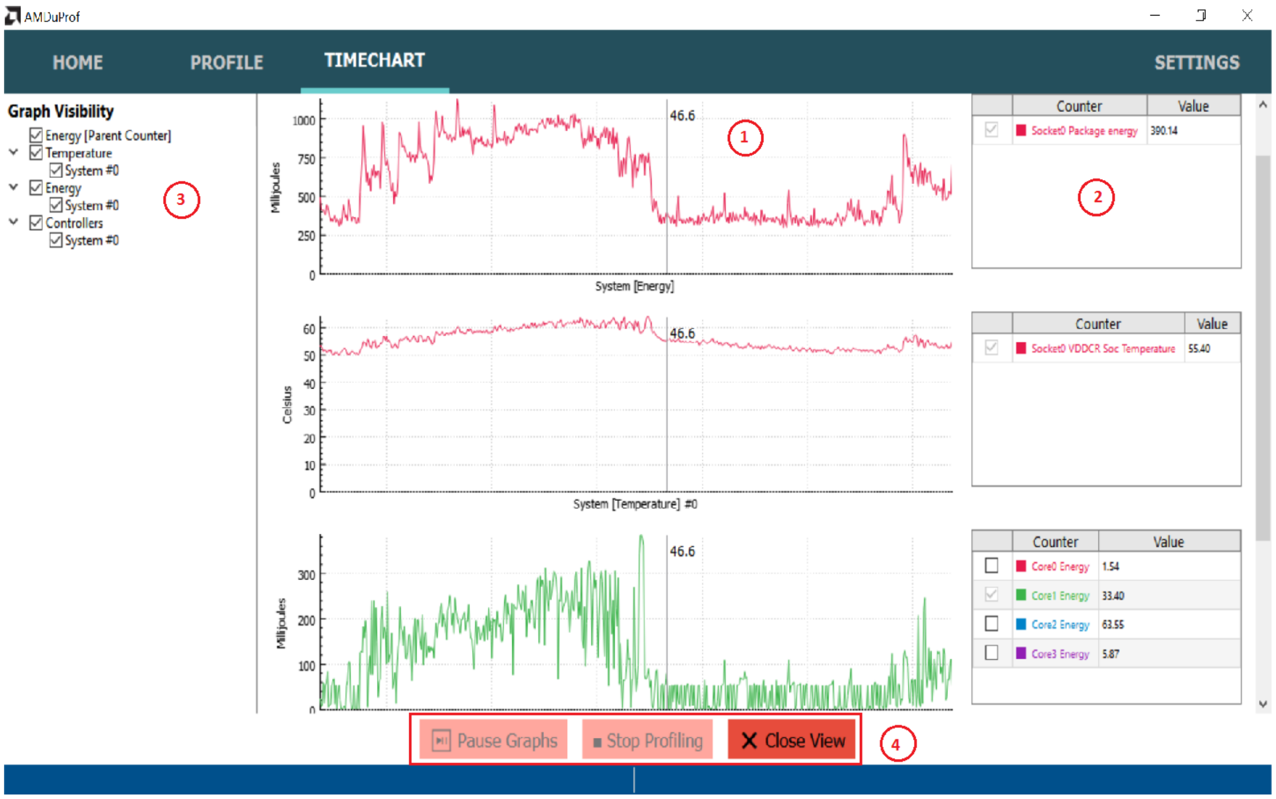
Start Profiling – Select Profile Type (Live Power Profile)

1. Select profile type as **System-wide Power Profile (Live)** from the drop-down list. This will list all the supported counter categories.
2. Clicking on an interesting counter category, will list the components for which this counter is selected as a tree selection.
3. Enable the interesting counters from this counter tree. Multiple counter categories can be configured
4. Show live graphs while profile is running? option lets you render the graphs live during profiling or save the data in database(.db file) during profiling and render the graphs after the profile data collection completed.

Once all the options are set correctly and clicking the **Start Profile** button will start the profile data collection. In this profile type, the profile data will be reported as line graphs in the **TIMECHART** page for further analysis.

6.2.2 Analyze

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.



	Counter	Value
✓	🟥 Socket0 Package energy	390.14

	Counter	Value
✓	🟥 Socket0 VDDCR Soc Temperature	55.40

	Counter	Value
☐	🟥 Core0 Energy	1.54
✓	🟩 Core1 Energy	33.40
☐	🟦 Core2 Energy	63.55
☐	🟪 Core3 Energy	5.87

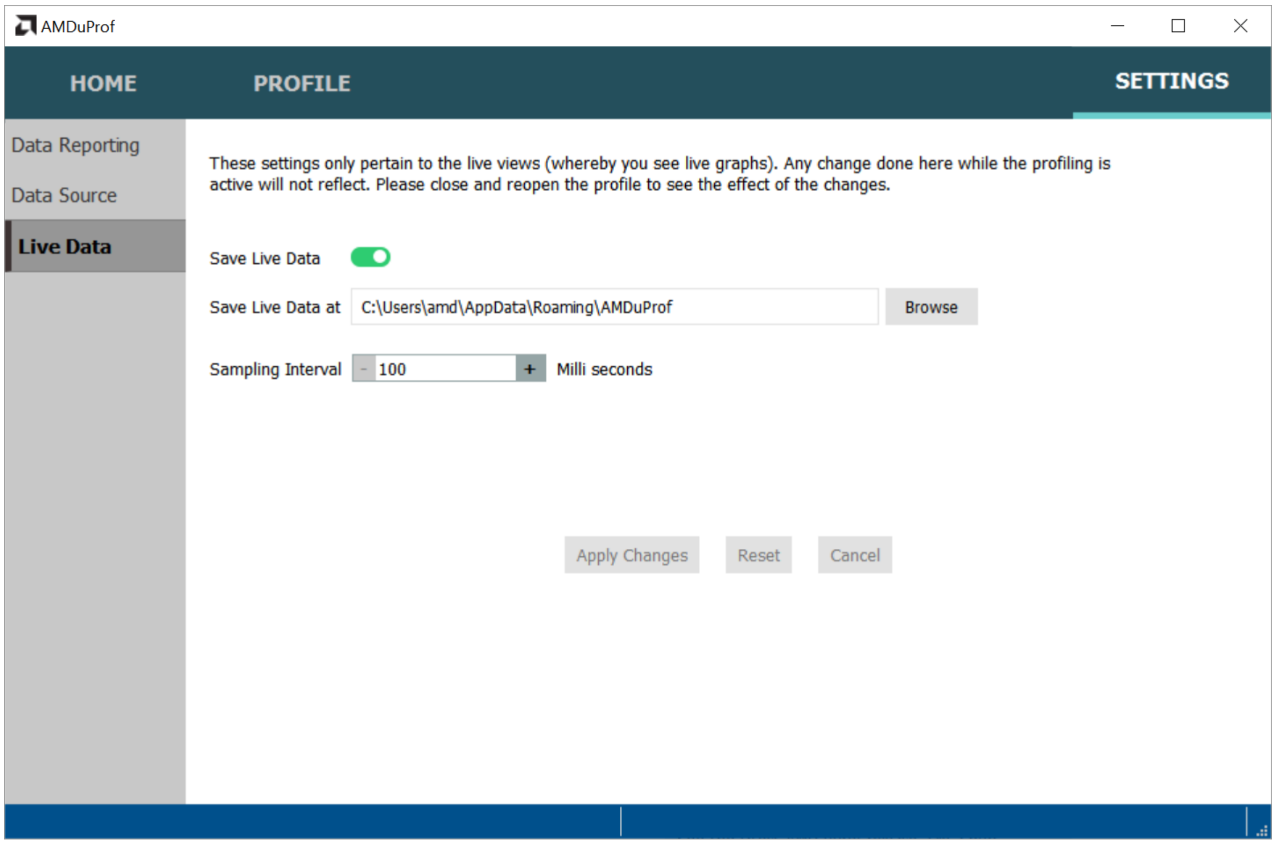
TIMECHART page – timeline graphs

1. In the **TIMECHART** page the metrics will be plotted in the live timeline graphs. Line graphs are grouped together and plotted based on the category.
2. There is also a corresponding data table adjacent to each graph to display the current value of the counters.
3. Graph Visibility pane on the left vertical pane will let you choose the graph to display.
4. When plotting is in progress various buttons are available, to let you
   - 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

6.2.3 Settings

The **SETTINGS → Live Data** window lets you to select whether you want to save to DB and specify the path in which the profile data DB can be stored. Also, the sampling interval too can be modified.

Note: This settings changes should be done before you start the profiling.



These settings only pertain to the live views (whereby you see live graphs). Any change done here while the profiling is active will not reflect. Please close and reopen the profile to see the effect of the changes.

Save Live Data

Save Live Data at C:\Users\amd\AppData\Roaming\AMDuProf

Sampling Interval 100 Milli seconds

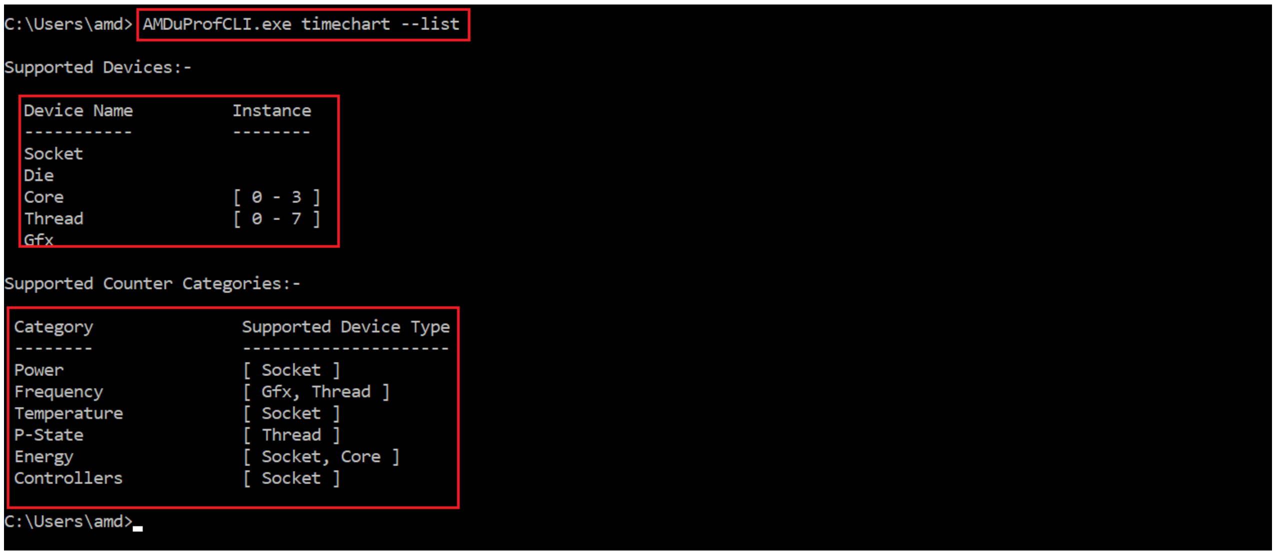
SETTINGS – Live Data

6.3 Profile using CLI

AMDuProfCLI's **timechart** command lets you collect the system metrics and write them into a text file or comma-separated-value (CSV) file. To collect power profile counter values, you need to follow these steps:

1. Get the list of supported counter categories by running AMDuProfCLI's **timechart** command with **--list** option
2. Collect and the report the required counters using AMDuProfCLI's timechart command by specifying the interesting counters with **-e** or **--event** option

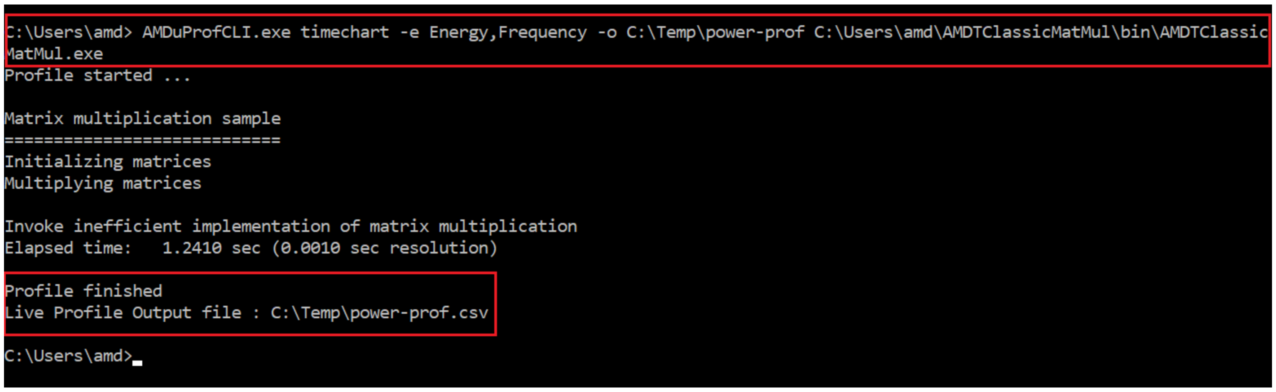
The timechart run to list the supported counter categories:



```
C:\Users\amd> AMDuProfCLI.exe timechart --list

Supported Devices:-

 Device Name          Instance
 -----------          --------
 Socket
 Die
 Core                 [ 0 - 3 ]
 Thread               [ 0 - 7 ]
 Gfx

Supported Counter Categories:-

 Category                Supported Device Type
 --------                ---------------------
 Power                   [ Socket ]
 Frequency               [ Gfx, Thread ]
 Temperature             [ Socket ]
 P-State                 [ Thread ]
 Energy                  [ Socket, Core ]
 Controllers             [ Socket ]

C:\Users\amd>_
```

AMDuProfCLI timechart --list command's output

The timechart run to collect the profile samples and write into a file:



```
C:\Users\amd> AMDuProfCLI.exe timechart -e Energy,Frequency -o C:\Temp\power-prof C:\Users\amd\AMDTClassicMatMul\bin\AMDTClassic
MatMul.exe
Profile started ...

Matrix multiplication sample
============================
Initializing matrices
Multiplying matrices

Invoke inefficient implementation of matrix multiplication
Elapsed time:   1.2410 sec (0.0010 sec resolution)

Profile finished
Live Profile Output file : C:\Temp\power-prof.csv

C:\Users\amd>_
```

AMDuProfCLI timechart run

The above run will collect the energy and frequency counters on all the devices on which these counters are supported and writes them in the output file specified with -o option. Before the profiling begins, the given application will be launched, and the data will be collected till the application terminates.

6.3.1 Examples

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
  ```

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
  ```

6.4 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

6.4.1 How to use the APIs?

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.

A sample program `CollectAllCounters.cpp` that uses these APIs, is available at `<AMDuProf-install-dir>/Examples/CollectAllCounters/` dir. To build and execute the example application, following steps should be performed:

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.

Linux

- To build

```
$ 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 execute

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

6.5 Limitations

- 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 7 Energy Analysis

Power Application Analysis

AMD uProf profiler offers Power Application Analysis to identify energy hotspots in the application. This is **Windows OS** only functionality. This profile type is used to analyze the energy consumption of an application or processes running in the system.

Features

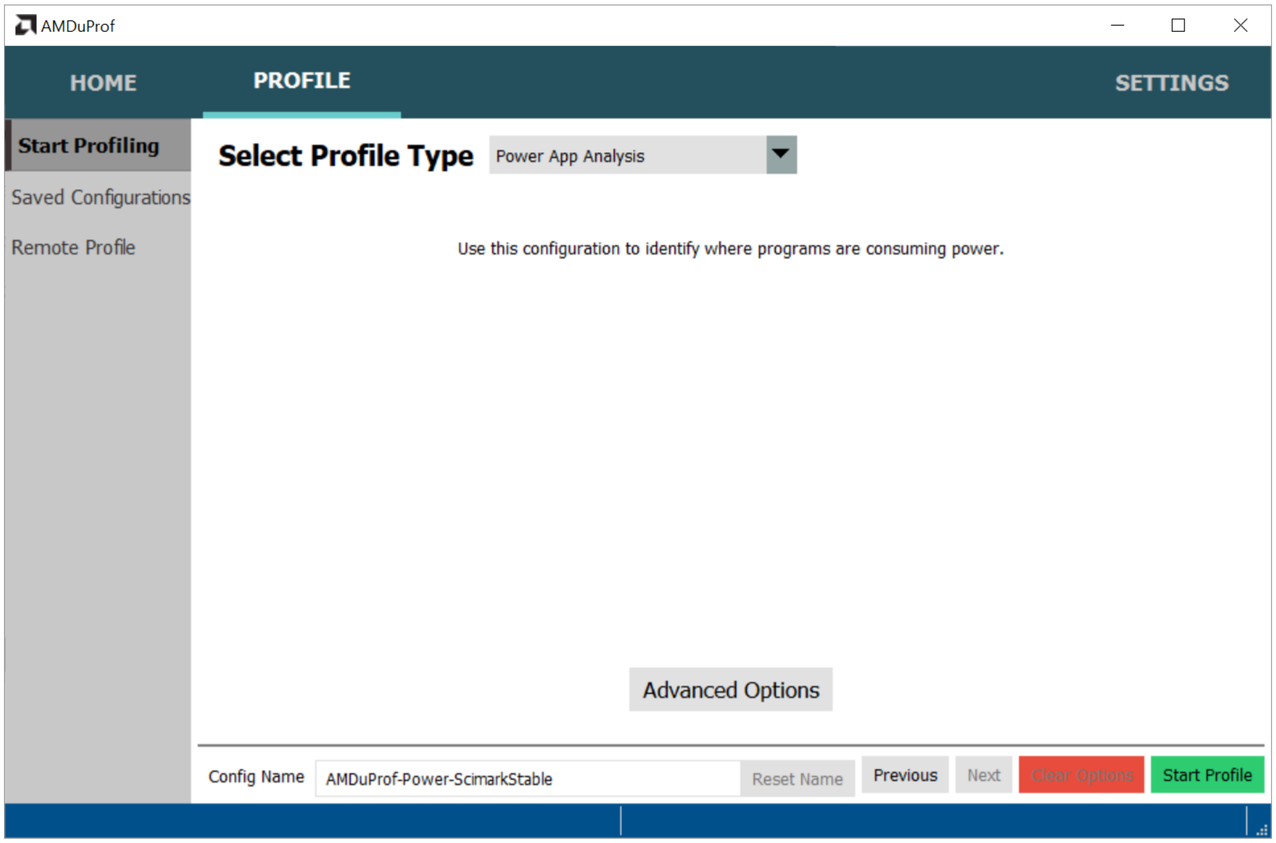
- Profile data
    - Periodically RAPL core energy values are sampled using OS timer as sampling event
- Profile mode
    - Profile data is collected when the application is running in user and kernel mode
- Profiles
    - C, C++, FORTRAN, Assembly applications
    - Various software components – Applications, Dynamically linked/loaded modules and OS kernel modules
- Profile data is attributed at various granularities
    - Process / Thread / Load Module / Function / Source line
    - To correlate the profile data to Function and Source line, debug information emitted by the compiler is required
- 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.
- AMDuProf GUI has various UIs to analyze and view the profile data at various granularities
    - Hot spots summary
    - Process and function analysis
    - Source and disassembly analysis

7.1 Profile using GUI

Here are the steps to configure and analyze the profile data:

7.1.1 Configure and Start profile

1. 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. You will see **Select Profile Target** fragment in the **Start Profiling** window. After selecting the appropriate profile target, clicking **Next** button will take you to **Select Profile Type** fragment.
3. In **Select Profile Type** fragment selecting **Power App Analysis** from the drop-down list, will take you to the below screenshot.



Select Profile Type: Power App Analysis

Use this configuration to identify where programs are consuming power.

Config Name: AMDuProf-Power-ScimarkStable

Power App Analysis - Configure

4. Once all the options are set, the **Start Profile** button at the bottom will be enabled and you can click on it to start the profile. After the profile initialization you will see *this* profile data collection screen.

To Analyze the profile data

5. When the profiling stopped, the collected raw profile data will be processed automatically, and you will the **Hot spots** window of **SUMMARY** page. Refer *this* section for more information on this window.

6. Click on the **ANALYZE** button on the top horizontal navigation bar to go **Metrics** window, which displays the profile data table at various granularities - Process, Load Modules, Threads and Functions. Refer *this* section for more information on this window.

7. Double-clicking any entry on the **Functions** table in **Metrics** window will make the GUI load the source tab for that function in **SOURCES** page. Refer *this* section for more information on this window

7.2 Profile using CLI

To profile and analyze the performance of a native (C/C++) application, you need to follow these steps:

1. Prepare the application. Refer *section* on how to prepare an application for profiling
2. Collect the samples for the application using AMDuProfCLI's **collect** command
3. Generate the report using AMDuProfCLI's **report** command, in readable format for analysis

Preparing the application is to build the launch application with debug information as debug info is needed to correlate the samples to functions and source lines.

The collect command will launch the application (if given) and collect the profile data and will generate raw data file (**.pdata** on Windows) and other miscellaneous files.

The report command translates the collected raw profile data to aggregate and attribute to the respective processes, threads, load modules, functions and instructions and writes them into a DB and then generate a report in CSV format.

Example

- Launch classic.exe and collect energy samples for that launch application:

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

- Generate report from the raw .pdata datafile:

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

- Generate report from raw .pdata file and use Symbol Server paths to resolve symbols:

```
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
```

7.3 Limitations

- Only one energy analysis profile session can run at a time.
- This is Windows OS only feature

Chapter 8 Remote Profiling

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

Supported configurations:-

- Host OS: Windows, Linux
- Target OS: Windows, Linux

8.1 Profile remote targets using GUI

8.1.1 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.

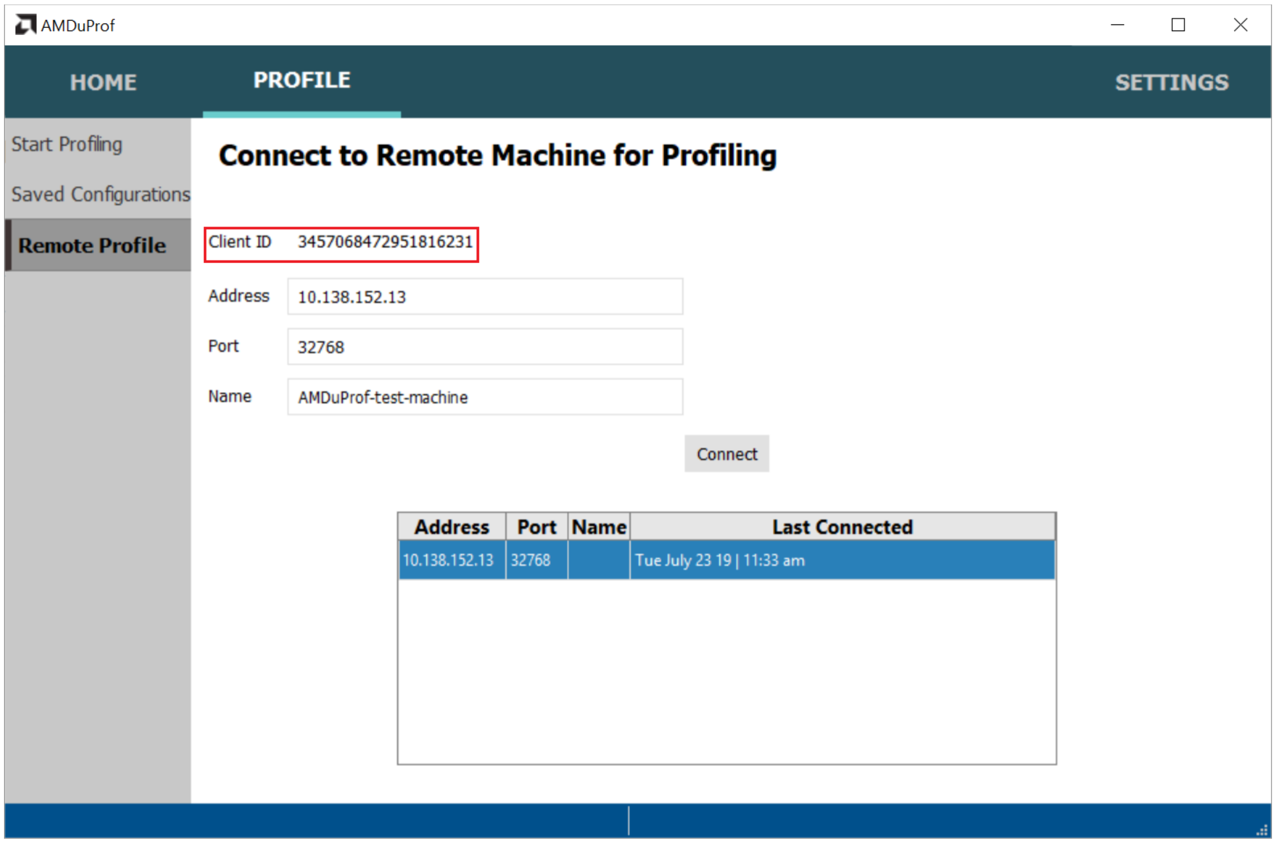
  To generate unique uid using AMDuProfCLI

  ```
  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
  ```



Connect to Remote Machine for Profiling

Client ID 3457068472951816231

Address 10.138.152.13

Port 32768

Name AMDuProf-test-machine

Address	Port	Name	Last Connected
10.138.152.13	32768		Tue July 23 19 | 11:33 am

Unique Client ID

8.1.2 Launching Remote Agent

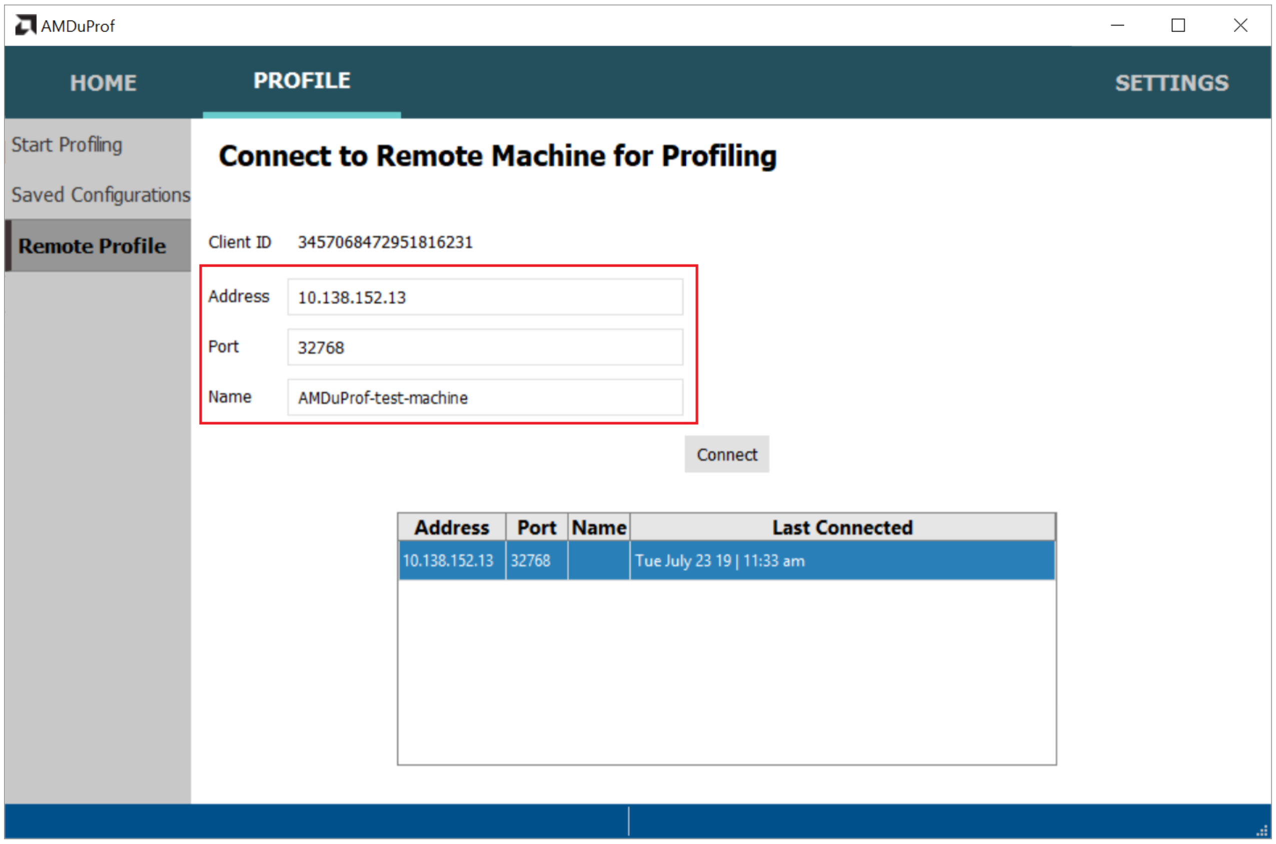
The uProf remote agent **AMDRemoteAgent** runs on the remote target system allows AMD uProf clients installed on other machines to connect to that remote system and execute Performance and Power profiling sessions of applications running on that remote system.

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...
```

8.1.3 Establishing connection with Remote Agent

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.



Connect to Remote Machine for Profiling

Client ID 3457068472951816231

Address 10.138.152.13

Port 32768

Name AMDuProf-test-machine

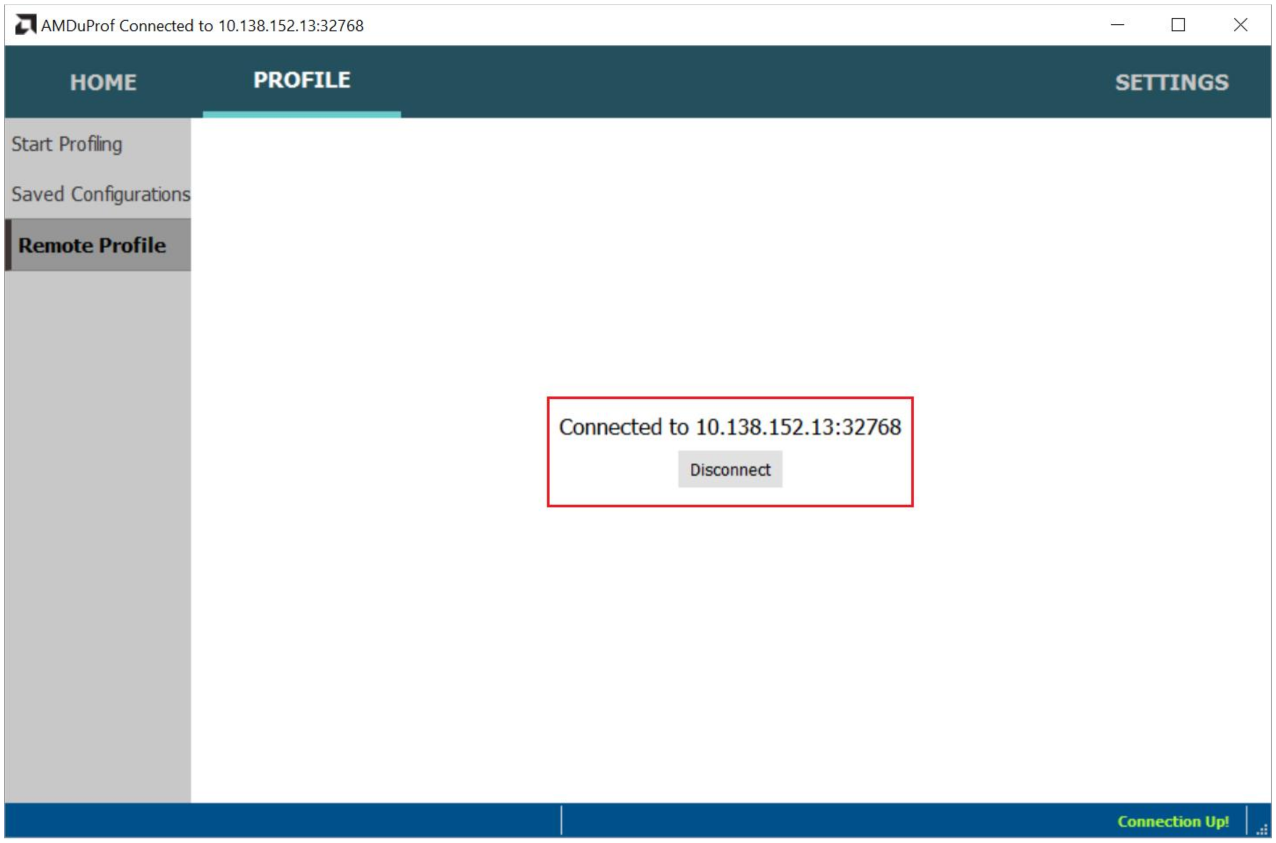
Address	Port	Name	Last Connected
10.138.152.13	32768		Tue July 23 19 | 11:33 am

Establishing remote connection

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. To disconnect, you need to navigate to **Remote Profile** window of **PROFILE** page and click **Disconnect** button as shown in the below screenshot.



Disconnecting remote connection

8.2 Profile remote targets using CLI

Following steps are to be followed to collect profile data from a remote target system

1. Generate user-id in the host system.

   Before establishing a connection with the remote agent, the user has to add the unique UID generated in the host client system. To generate unique uid, run the following command on client system

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

2. Add this uid to remote agent running on the remote target system, by running

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

3. Launch remote agent binary **AMDRemoteAgent** on the remote target system - which allows AMD uProf clients installed on other machines to connect to that remote system and collect profile data.

```
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...
```

4. Run AMDuProfCLI commands from the client system using --ip and --port option to profile on that remote target system

```
C:\> AMDuProfCLI.exe collect --config assess -o c:\Temp\cpuprof-assess --
ip 127.0.0.1 –port 27016 AMDTClassicMatMul.exe

C:\> AMDuProfCLI.exe report -i c:\Temp\cpuprof-assess -o c:\Temp\cpuprof-
assess\ --ip 127.0.0.1 –port 27016

C:\> AMDuProfCLI.exe timechart --event core=0-3,frequency --output
C:\Temp\power_output.txt --duration 10 --format txt  --ip  127.0.0.1  –port
27016
```

8.3 Limitations

- Only one instance of GUI or CLI client process for a user (having unique client id) can establish connection with AMDRemoteAgent process running on the target system.
- Multiple GUI or CLI client processes with different unique client ids (from same or different host client systems), can establish connection with the AMDRemoteAgent process running on the target system.
- The AMDRemoteAgent process can entertain either CPU or Power profile session at a time from a client process.
- The AMDRemoteAgent process can entertain CPU profile request from one client process and Power profile request from another client process simultaneously.

Chapter 9 Profile Control APIs

9.1 AMDProfileControl APIs

The AMDProfileControl APIs allow you to limit the profiling scope to a specific portion of the code within the target application.

Usually while profiling an application, samples for the entire control flow of the application execution will be collected - i.e. from the start of execution till end of the application execution. The control APIs can be used to enable the profiler to collect data 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 instrumenting the application to enable/disable profiling of the interesting code regions only.

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:

```
<AMDuProf-install-dir>\lib\x86\AMDProfileController.lib
<AMDuProf-install-dir>\lib\x64\AMDProfileController.lib
```

Linux:

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

9.1.1 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

```
bool amdProfileResume (AMD_PRPOFILE_CPU);
```

amdProfilePause

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

```
bool amdProfilePause (AMD_PRPOFILE_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.

9.1.2 How to use 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.

```
#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 ();

     // Stop the CPU Profile data collection
     amdProfilePause (AMD_PROFILE_CPU);

     return 0;
}
```

9.1.3 Compiling instrumented target application

Windows

To compile the application on Microsoft Visual Studio, update the configuration properties to include the path of header file and link with `AMDProfileController.lib` library.

Linux

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

```
$ 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++.

9.1.4 Profiling instrumented target application

AMDuProf GUI

After compiling the target application, create a profile configuration in AMDuProf 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.

AMDuProfCLI

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

Windows:

```
C:\> AMDuProfCLI.exe collect --config tbp --start-paused -o C:\Temp\prof-tbp
ClassicCpuProfileCtrl.exe
```

Linux:

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

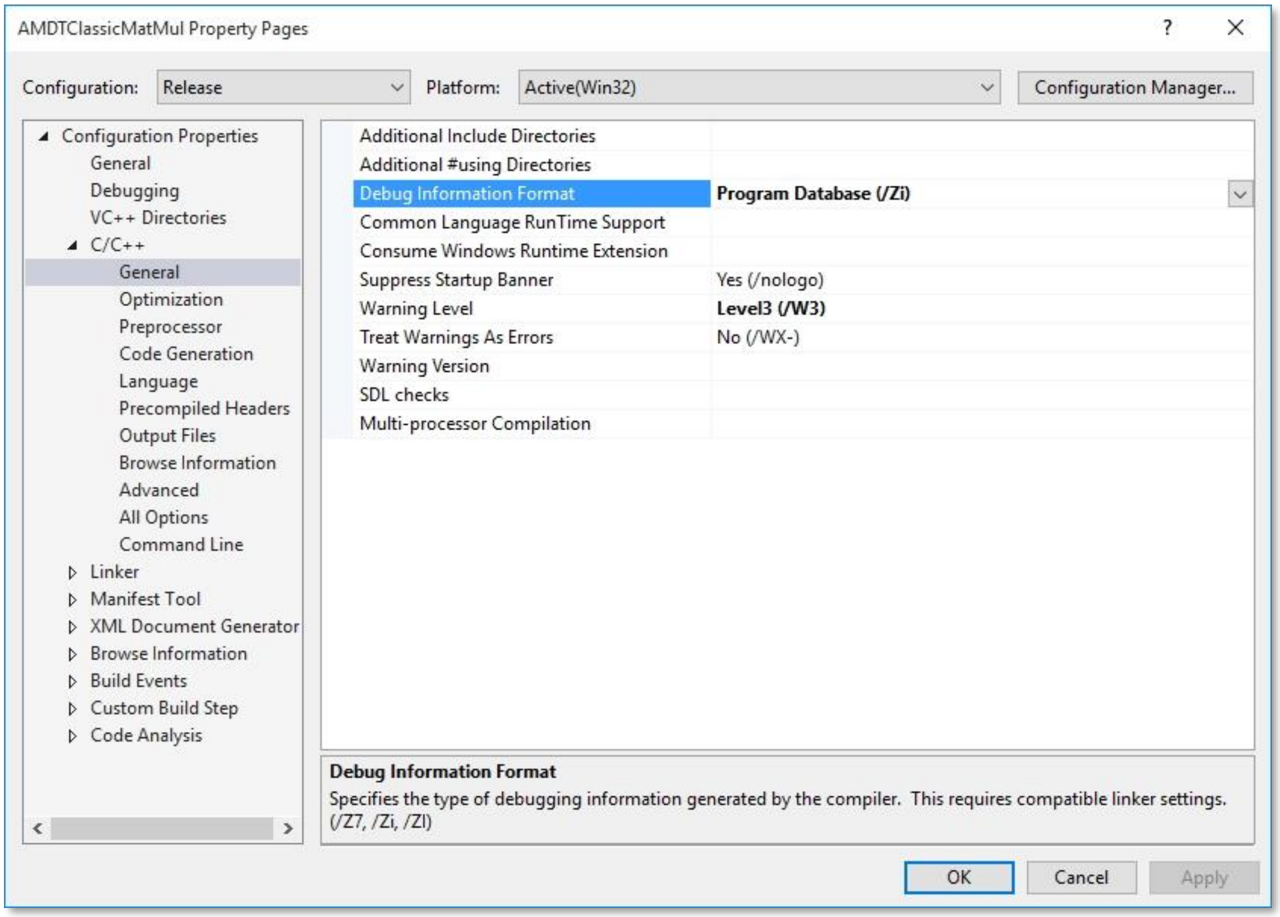
Chapter 10 Reference

10.1 Preparing an application for profiling

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

10.1.1 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 x64 application in release mode:


AMDTClassicMatMul Property Pages

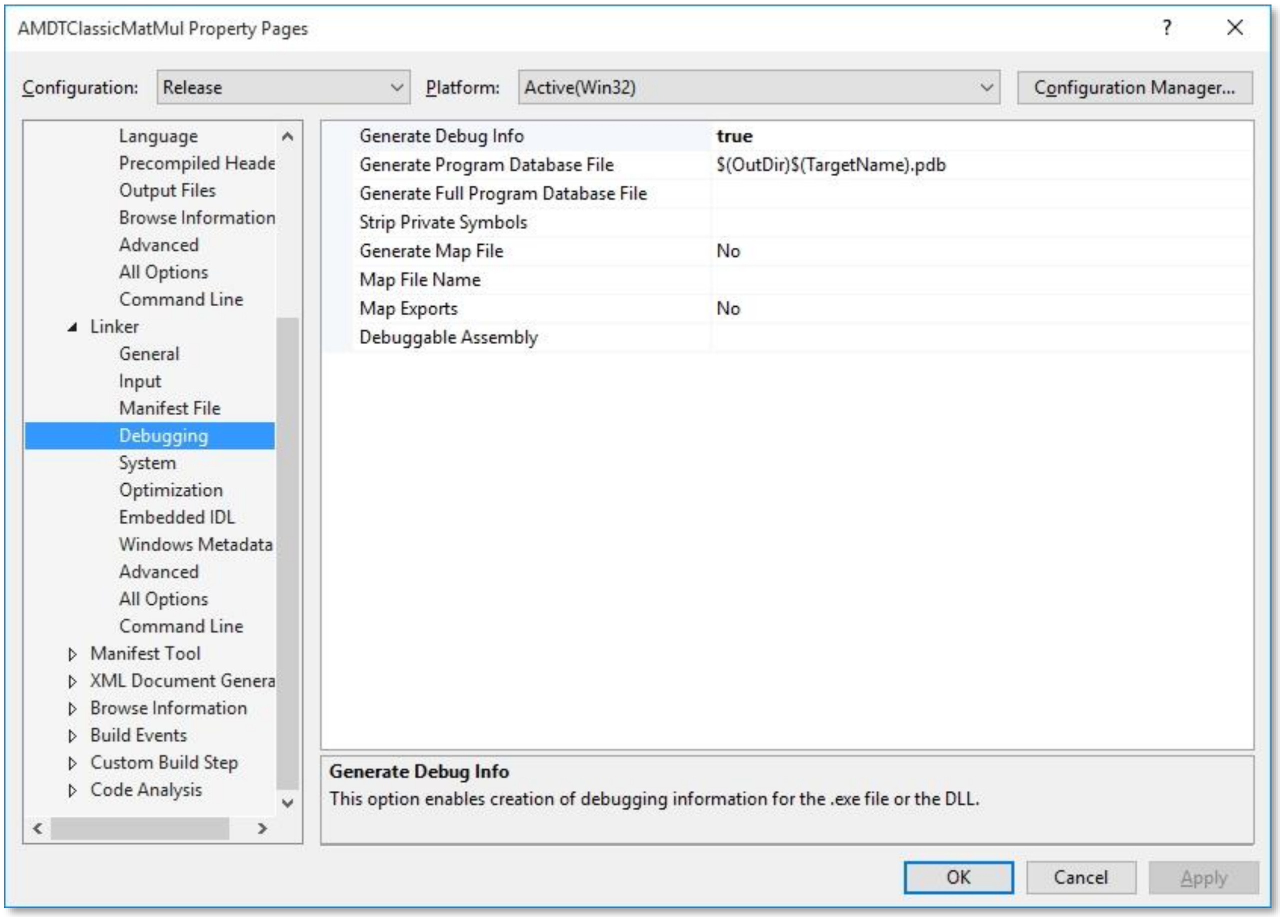
Configuration: Release
Platform: Active(Win32)

Additional Include Directories	
Additional #using Directories	
Debug Information Format	Program Database (/Zi)
Common Language RunTime Support	
Consume Windows Runtime Extension	
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level3 (/W3)
Treat Warnings As Errors	No (/WX-)
Warning Version	
SDL checks	
Multi-processor Compilation	

Debug Information Format
Specifies the type of debugging information generated by the compiler. This requires compatible linker settings. (/Z7, /Zi, /ZI)

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 and 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)**.



Configuration: Release   Platform: Active(Win32)

Generate Debug Info	true
Generate Program Database File	$(OutDir)$(TargetName).pdb
Generate Full Program Database File	
Strip Private Symbols	
Generate Map File	No
Map File Name	
Map Exports	No
Debuggable Assembly	

Generate Debug Info
This option enables creation of debugging information for the .exe file or the DLL.

6. In the project pane, expand the ‘Linker’ item; then select the ‘Debugging’ item.
7. In the ‘Generate Debug Info’ list, select (/DEBUG).

10.1.2 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.

10.2 CPU 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.

10.2.1 Hardware Sources

Performance Monitor Counters (PMC)

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

- In counting mode, these counters are used to count the specific events that occur in a CPU core.
- In sampling mode, these counters are programmed to count a specific number of events. Once the count is reached the appropriate number of times (called sampling interval), an interrupt is triggered. During the interrupt handling, the CPU Profiler collects profile data.

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

Instruction-Based Sampling (IBS)

IBS is a code profiling mechanism that enables the processor to select a random instruction fetch or micro-Op after a programmed time interval has expired and record specific performance information about the operation. An interrupt is generated when the operation is complete as specified by IBS Control MSR. An interrupt handler can then read the performance information that was logged for the operation.

The IBS mechanism is split into two parts:
- Instruction Fetch performance
- Instruction Execution Performance

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

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

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

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

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

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

L3 Cache Performance Monitor Counters (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.

10.2.2 Profiling Concepts

Sampling

Sampling profilers works 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.

Sampling point: 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 by the interrupt handler.

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.

10.2.3 Profile Types

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

Time-Based Profile (TBP)

In this profile, the profile data is periodically collected based on the specified 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.

10.2.4 CPU PMC Events

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

Core PMC Events

PMC Event	Description
[PMCx076] CPU clock cycles not halted	The number of core clocks that the CPU is not in a halted state.
[PMCx0C0] Retired Instructions	The number of instructions retired
[PMCx0C1] Retired uops	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
[PMCx0C2] Retired Branch Instructions	The number of branch instructions retired. This includes all types of architectural control flow changes, including exceptions and interrupts.
[PMCx0C3] Retired Branch Instructions Mispredicted	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).

[PMCx0C4] Retired Taken Branch Instructions	The number of taken branches that were retired. This includes all types of architectural control flow changes, including exceptions and interrupts.
[PMCx0CA] Retired Indirect Branch Instructions Mispredicted	Retired Indirect Branch Instructions Mispredicted
[PMCx08A] L1 BTB Correction	L1 BTB Correction
[PMCx08B] L2 BTB Correction	L2 BTB Correction
[PMCx040] Data Cache Accesses	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.
[PMCx041] MAB Allocation by Pipe	MAB allocation by pipe
[PMCx043] Data Cache Refills from System	Demand Data Cache Fills by Data Source
[PMCx045] L1 DTLB Miss	L1 DTLB Miss
[PMCx047] Misaligned loads	Misaligned loads (accesses).
[PMCx080] 32 Byte Instruction Cache Fetches	The number of 32B fetch windows transferred from IC pipe to DE instruction decoder (includes non-cacheable and cacheable fill responses)
[PMCx081] 32 Byte Instruction Cache Misses	The number of 32B fetch windows tried to read the L1 IC and missed in the full tag.
[PMCx084] L1 ITLB Miss, L2 ITLB Hit	The number of instruction-fetches that miss in the L1 ITLB but hit in the L2 ITLB
[PMCx085] L1 ITLB Miss, L2 ITLB Miss	The number of instruction-fetches that miss in both the L1 and L2 TLBs.

CPU Performance Metrics

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

10.2.5 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

IBS Fetch Event	Description
All IBS fetch samples	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
IBS fetch killed	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.)
IBS fetch attempted	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.
IBS fetch completed	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.)
IBS fetch aborted	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.
IBS ITLB hit	The number of IBS attempted fetch samples where the fetch operation initially hit in the L1 ITLB (Instruction Translation Lookaside Buffer).
IBS L1 ITLB misses (and L2 ITLB hits)	The number of IBS attempted fetch samples where the fetch operation initially missed in the L1 ITLB and hit in the L2 ITLB.
IBS L1 L2 ITLB miss	The number of IBS attempted fetch samples where the fetch operation initially missed in both the L1 ITLB and the L2 ITLB.
IBS instruction cache misses	The number of IBS attempted fetch samples where the fetch operation initially missed in the IC (instruction cache).
IBS instruction cache hit	The number of IBS attempted fetch samples where the fetch operation initially hit in the IC.

IBS 4K page translation	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.
IBS 2M page translation	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.
IBS fetch latency	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.
IBS fetch L2 cache miss	The instruction fetch missed in the L2 Cache.
IBS ITLB refill latency	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

IBS Op Event	Description
All IBS op samples	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.
IBS tag-to-retire cycles	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.
IBS completion-to-retire cycles	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.
IBS branch op	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.

IBS mispredicted branch op	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.
IBS taken branch op	The number of IBS samples for retired branch operations that were taken branches.
IBS mispredicted taken branch op	The number of IBS samples for retired branch operations that were mispredicted taken branches.
IBS return op	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.
IBS mispredicted return op	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.
IBS resync op	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.
IBS all load store ops	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.
IBS load ops	The number of IBS op samples for ops that perform a load operation.
IBS store ops	The number of IBS op samples for ops that perform a store operation.
IBS L1 DTLB hit	The number of IBS op samples where either a load or store operation initially hit in the L1 DTLB (data translation lookaside buffer).
IBS L1 DTLB misses L2 hits	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.

IBS L1 and L2 DTLB misses	The number of IBS op samples where either a load or store operation initially missed in both the L1 DTLB and the L2 DTLB.
IBS data cache misses	The number of IBS op samples where either a load or store operation initially missed in the data cache (DC).
IBS data cache hits	The number of IBS op samples where either a load or store operation initially hit in the data cache (DC).
IBS misaligned data access	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).
IBS bank conflict on load op	The number of IBS op samples where either a load or store operation caused a bank conflict with a load operation.
IBS bank conflict on store op	The number of IBS op samples where either a load or store operation caused a bank conflict with a store operation.
IBS store-to-load forwarded	The number of IBS op samples where data for a load operation was forwarded from a store operation.
IBS store-to-load cancelled	The number of IBS op samples where data forwarding to a load operation from a store was cancelled.
IBS UC memory access	The number of IBS op samples where a load or store operation accessed uncacheable (UC) memory.
IBS WC memory access	The number of IBS op samples where a load or store operation accessed write combining (WC) memory.
IBS locked operation	The number of IBS op samples where a load or store operation was a locked operation.
IBS MAB hit	The number of IBS op samples where a load or store operation hit an already allocated entry in the Miss Address Buffer (MAB).
IBS L1 DTLB 4K page	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.
IBS L1 DTLB 2M page	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.

IBS L1 DTLB 1G page	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.
IBS L2 DTLB 4K page	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.
IBS L2 DTLB 2M page	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.
IBS L2 DTLB 1G page	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.
IBS data cache miss load latency	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.
IBS load resync	Load Resync.
IBS Northbridge local	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.
IBS Northbridge remote	The number of IBS op samples where a load operation was serviced from a remote processor.
IBS Northbridge local L3	The number of IBS op samples where a load operation was serviced by the local L3 cache.
IBS Northbridge local core L1 or L2 cache	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.
IBS Northbridge local core L1, L2, L3 cache	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.

IBS Northbridge local DRAM	The number of IBS op samples where a load operation was serviced by local system memory (local DRAM via the memory controller).
IBS Northbridge remote DRAM	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).
IBS Northbridge local APIC MMIO Config PCI	The number of IBS op samples where a load operation was serviced from local MMIO, configuration or PCI space, or from the local APIC.
IBS Northbridge remote APIC MMIO Config PCI	The number of IBS op samples where a load operation was serviced from remote MMIO, configuration or PCI space.
IBS Northbridge cache modified state	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.
IBS Northbridge cache owned state	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.
IBS Northbridge local cache latency	The total data cache miss latency (in processor cycles) for load operations that were serviced by the local processor.
IBS Northbridge remote cache latency	The total data cache miss latency (in processor cycles) for load operations that were serviced by a remote processor.

10.3 Useful links

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

AMD Developer Documents: *https://developer.amd.com/resources/developer-guides-manuals/*

Open Source Register Reference (OSRR) for AMD Family 17h Processors:
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

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