AMD uProf User Guide

Version 3.4

Advanced Micro Devices

Trademarks

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
AMDuProfPcm	Denotes the CLI that is used to perform system analysis
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 AMDuProfPcm

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 functionalities to perform:

- **Performance Analysis** (CPU Profile)
  - To identify runtime performance bottlenecks of the application
- **System Analysis**
  - To monitor basic system performance metrics like IPC, memory bandwidth
- **Live Power Profile**
  - To monitor thermal and power characteristics of the system
- **Energy Analysis**
  - To identify energy hotspots in the application (Windows only)

AMD uProf has following user interfaces:

Executable	Description	Supported OS
AMDuProf	GUI to perform CPU & Power Profile	Windows, Linux
AMDuProfCLI	CLI to perform CPU & Power Profile	Windows, Linux, FreeBSD
AMDuProfPcm	CLI to perform System Analysis	Windows, Linux, FreeBSD
AMDRemoteAgent	CLI agent for remote profiling	Windows, Linux

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
- Parallelism
  - OpenMP
  - MPI
- 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

The latest version of the AMD uProf installer package for the supported Operating systems can be downloaded from *https://developer.amd.com/amd-uprof/*. Install AMD uProf using one of the following methods.

1.3.1 Windows

Run the 64-bit Windows installer binary `AMDuProf-x.y.z.exe`. Upon successful completion of the installation the executables, libraries and the other required files will be installed at `C:\Program Files\AMD\AMDuProf\` folder.

1.3.2 Linux

Install using tar file

Install uProf from the downloaded tar file, by extracting the tar.bz2 binary.

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

The Power Profiler Linux driver must be installed manually. To do that, refer *this* section.

Install using RPM package (RHEL)

Install the uProf RPM package by either using the **rpm** or **yum** command.

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

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

Upon successful completion of the installation the executables, libraries and the other required files will be installed at `/opt/AMDuProf_X.Y-ZZZ/` directory.

Install using Debian package (Ubuntu)

Install the uProf Debian package by using the **dpkg** command.

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

Upon successful completion of the installation the executables, libraries and the other required files will be installed at `/opt/AMDuProf_X.Y-ZZZ/` directory.

Installing Power Profiling driver on Linux

While installing uProf using RPM and Debian installer packages, the Power Profiling driver gets build and installed automatically. However, if you have downloaded the AMD uProf tar.bz2 archive, you must install the Power Profiler's Linux driver manually.

The GCC and MAKE software packages are prerequisites for installing Power Profiler's Linux driver. If you do not 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
```

Perform the following steps:

```
$ 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 in this directory and 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.

1.3.3 FreeBSD

Install using tar file

Install uProf from the downloaded tar file, by extracting the tar.bz2 binary.

```
$ tar -xf AMDuProf_FreeBSD_x64_x.y.z.tar.bz2
```

1.4 Sample programs

Few sample programs are installed along with the product is installed along with the product to let you use with the tool.

Windows:

- A sample matrix multiplication application

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

Linux:

- A sample matrix multiplication program with makefile

  `/opt/AMDuProf_X.Y-ZZZ/Examples/AMDTClassicMat/`

- An OpenMP example program and its variants with makefile

  `/opt/AMDuProf_X.Y-ZZZ/Examples/CollatzSequence_C-OMP/`

FreeBSD:

- A sample matrix multiplication program with makefile

  `/<install dir>/AMDuProf_FreeBSD_x64_X.Y.ZZZ/Examples/AMDTClassicMat/`

1.5 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/t5/server-gurus/ct-p/amd-server-gurus*

Chapter 2 CPU Profiling - workflow and key concepts

2.1 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
  - HPC - to analyze OpenMP profile data
  - Cache Analysis - to analyze the hot cache lines that are false shared
- Profile Control API to selectively enable and disable profiling from the target application by instrumenting it, to limit the scope of the profiling

2.2 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.

2.2.1 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) OpenMP Trace – OMPT (Linux)	OS Timer, Core PMC events, IBS

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.

2.2.2 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.

2.2.3 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.

2.3 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.

2.4 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 AMDuProfPcm – System Analysis

System Analysis utility AMDuProfPcm helps to monitor basic performance monitoring metrics for AMD's family 17h processors. This utility periodically collects the CPU Core, L3 & DF performance events count values and report various metrics.

Notes:

- This tool is supported on Windows, Linux, and FreeBSD.
- On Linux:
    - AMDuProfPcm uses **msr** driver and either requires root privileges or read write permissions for /dev/cpu/*/msr devices.
    - NMI watchdog needs to be disabled. (echo 0 > /proc/sys/kernel/nmi_watchdog)
- On FreeBSD, AMDuProfPcm uses **cpuctl** module and either requires root privileges or read write permissions for /dev/cpuctl* devices

Synopsis:

```
AMDuProfPcm [<OPTIONS>] -- [<PROGRAM>] [<ARGS>]
```

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

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

Common usages:

```
$ AMDuProfPcm -h

# AMDuProfPcm -m ipc -c core=0 -d 10 -o /tmp/pmcdata.txt

# AMDuProfPcm -m memory -a -d 10 -o /tmp/memdata.txt -- /tmp/myapp.exe
```

Options:

Option	Description
`-h`	Displays this help information on the console/terminal.
`-m <metric,...>`	Metrics to report. Default metric group is 'ipc'. Supported metric groups and the corresponding metrics are Platform, OS, and Hypervisor specific.

	Run “AMDuProfpcm -h” to get the list of supported metrics. In general, following metric groups will be supported: **ipc** – reports metrics like CEF, Utilization, CPI, IPC **fp** – reports GFLOPS **l1** – L1 cache related metrics (DC access and IC Fetch miss ratio) **l2** – L2D and L2I cache related access / hit / miss metrics **l3** – L3 cache metrics like L3 Access, L3 Miss, and Average Miss latency **dc** – advanced caching metrics like DC refills by source **memory** – approximate memory read and write bandwidths in GB/s for all the channels **pcie** – PCIe bandwidth in GB/s **xgmi** – approximate xGMI outbound data bytes in GB/s for all the remote links
`-c <core|ccx|ccd|package=<n>`	Collect from the specified core | ccx | die | package. Default is 'core=0'. If '**ccx**' is specified: - core events will be collected from all the cores of this ccx. - l3 events will be collected from the first core of this ccx. - df events will be collected from the first core of this ccx. If '**die**' is specified: - core events will be collected from all the cores of this die. - l3 events will be collected from the first core of all the ccx's of this die.

	- df events will be collected from the first core of this die. If '**package**' is specified: - core events will be collected from all the cores of this package. - l3 events will be collected from the first core of all the ccx's of this package. - df events will be collected from the first core of all the die of this package.
`-a`	Collect from all the cores. Note: Options -c and -a cannot be used together.
`-C`	Prints the cumulative data at the end of the profile duration. Otherwise, all the samples will be reported as timeseries data.
`-A <system,package,ccd,ccx,core>`	Print aggregated metrics at various component level. Following are various granularity that are supported: **system** – samples from all the cores in the system will be aggregated; **package** - samples from all the cores in the package will be aggregated and reported for all the packages available in the system; Applicable for multi-package systems. **ccd** - samples from all the cores in CCD will be aggregated and reported for all the CCDs. **ccx** - samples from all the cores in CCX will be aggregated and reported for all the CCXs. **core** - samples from all the cores on which samples are collected will be reported without aggregation. Note: - Option -a should be used along with this option to collect samples from all the cores. - Comma separated list of components can be specified

`-i <config file>`	User defined XML config file that specifies Core|L3|DF counters to monitor. Refer sample files at <install-dir>/bin/Data/Config/ dir for the format. Note: - Options -i and -m cannot be used together. - If option -i is used, all the events mentioned in the user-defined config file will be collected.
`-d <seconds>`	Profile duration to run
`-t < multiplex interval in ms>`	Interval in which PMC count values will be read. Minimum is 16ms
`-o <output file>`	Output file name. The output report will be in CSV format.
`-D <dump file>`	Output file that contains the event count dump for all the events that are being monitored. This output report will be in CSV format.
`-p <n>`	Set precision of the metrics reported. Default is 2.
`-q`	Hide CPU topology section in the output report.
`-r`	To force reset the MSRs
`-l`	List supported raw PMC events
`-z <pmc-event>`	Print the name, description, and available unit masks for the event.
`-x <core-id,...>`	Core affinity for launched application, comma separated list of core ids.
`-w <dir>`	Specify the working directory. Default will be the path of the launched application
`-v`	Print version

Following performance metrics are reported for AMD EPYC 2nd generation processors:

Metric group	Metric	Description
ipc	Utilization (%)	Percentage of time the Core was running – i.e., non-idle time
	Eff Freq	Core Effective Frequency (CEF) 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 Per Cycle (IPC) is the average number of instructions retired per cpu 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 (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.
	Branch Misprediction Ratio	The ration between mispredicted branches and retired branch instructions.
fp	Retired SSE/AVX Flops(GFLOPs)	The number of retired SSE/AVX FLOPs.
	Mixed SSE/AVX Stalls	Mixed SSE/AVX stalls. This metric is in per thousand instructions (PTI).
1l	IC(32B) Fetch Miss Ratio	Instruction cache fetch miss ratio.
	DC Access	All data cache (DC) accesses. This metric is in per thousand instructions (PTI)
l2	L2 Access	All L2 cache accesses. This metric is in per thousand instructions (PTI)

	L2 Access from IC Miss	L2 cache accesses from IC miss. This metric is in per thousand instructions (PTI)
	L2 Access from DC Miss	L2 cache accesses from DC miss. This metric is in per thousand instructions (PTI)
	L2 Access from HWPF	L2 cache accesses from L2 hardware prefetching. This metric is in per thousand instructions (PTI)
	L2 Miss	All L2 cache misses. This metric is in per thousand instructions (PTI)
	L2 Miss from IC Miss	L2 cache misses from IC miss. This metric is in per thousand instructions (PTI)
	L2 Miss from DC Miss	L2 cache misses from DC miss. This metric is in per thousand instructions (PTI)
	L2 Miss from HWPF	L2 cache misses from L2 hardware prefetching. This metric is in per thousand instructions (PTI)
	L2 Hit	All L2 cache hits. This metric is in per thousand instructions (PTI)
	L2 Hit from IC Miss	L2 cache hits from IC miss. This metric is in per thousand instructions (PTI)
	L2 Hit from DC Miss	L2 cache hits from DC miss. This metric is in per thousand instructions (PTI)
	L2 Hit from HWPF	L2 cache hits from L2 hardware prefetching. This metric is in per thousand instructions (PTI)
tlb	L1 ITLB Miss	The instruction fetches that misses in the L1 Instruction Translation Lookaside Buffer(ITLB) but hit in the L2-ITLB plus the ITLB reloads originating from page table walker. The table walk requests are made for L1-ITLB miss and L2-ITLB misses. This metric is in Per-Thousand-Instructions (PTI)
	L2 ITLB Miss	Number of ITLB reloads from page table walker due to L1-ITLB and L2-ITLB misses. This metric is in Per-Thousand-Instructions (PTI)

	L1 DTLB Miss	The number of L1 Data Translation Lookaside Buffer (DTLB) misses from load store micro-ops. This event counts both L2-DTLB hit and L2-DTLB miss. This metric is in Per-Thousand-Instructions (PTI)
	L2 DTLB Miss	The number of L2 Data Translation Lookaside Buffer (DTLB) missed from load store micro-ops. This metric is in Per-Thousand-Instructions (PTI)
l3	L3 Access	L3 cache accesses. This metric is in per thousand instructions (PTI)
	L3 Miss	L3 cache miss. This metric is in per thousand instructions (PTI)
	Ave L3 Miss Latency	Average L3 miss latency in core cycles.
memory	Mem Ch-A RdBw (GB/s) Mem Ch-A WrBw (GB/s) …	Memory Read and Write bandwidth in GB/s for all the memory channels.
xgmi	xGMI0 BW (GB/s) xGMI1 BW (GB/s) xGMI2 BW (GB/s) xGMI3 BW (GB/s)	Approximate xGMI outbound data bytes in GB/s for all the remote links.
pcie	PCIe0 (GB/s) PCIe1 (GB/s) PCIe2 (GB/s) PCIe3 (GB/s)	Approximate PCIe bandwidth in GB/s.

Following performance metrics are reported for AMD EPYC 3rd generation processors:

Metric group	Metric	Description
ipc	Utilization (%)	Percentage of time the Core was running – i.e., non-idle time
	Eff Freq	Core Effective Frequency (CEF) 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 Per Cycle (IPC) is the average number of instructions retired per cpu 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 (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.
	Branch Misprediction Ratio	The ration between mispredicted branches and retired branch instructions.
fp	Retired SSE/AVX Flops(GFLOPs)	The number of retired SSE/AVX FLOPs.
	Mixed SSE/AVX Stalls	Mixed SSE/AVX stalls. This metric is in per thousand instructions (PTI).
1l	IC(32B) Fetch Miss Ratio	Instruction cache fetch miss ratio.
	Op Cache (64B) Fetch Miss Ratio	Op Cache fetch miss ratio

	IC Access	All instruction cache accesses. This metric is in per thousand instructions (PTI)
	IC Miss	Instruction cache miss. This metric is in per thousand instructions (PTI)
	DC Access	All data cache (DC) accesses. This metric is in per thousand instructions (PTI)
l2	L2 Access	All L2 cache accesses. This metric is in per thousand instructions (PTI)
	L2 Access from IC Miss	L2 cache accesses from IC miss. This metric is in per thousand instructions (PTI)
	L2 Access from DC Miss	L2 cache accesses from DC miss. This metric is in per thousand instructions (PTI)
	L2 Access from HWPF	L2 cache accesses from L2 hardware prefetching. This metric is in per thousand instructions (PTI)
	L2 Miss	All L2 cache misses. This metric is in per thousand instructions (PTI)
	L2 Miss from IC Miss	L2 cache misses from IC miss. This metric is in per thousand instructions (PTI)
	L2 Miss from DC Miss	L2 cache misses from DC miss. This metric is in per thousand instructions (PTI)
	L2 Miss from HWPF	L2 cache misses from L2 hardware prefetching. This metric is in per thousand instructions (PTI)
	L2 Hit	All L2 cache hits. This metric is in per thousand instructions (PTI)
	L2 Hit from IC Miss	L2 cache hits from IC miss. This metric is in per thousand instructions (PTI)
	L2 Hit from DC Miss	L2 cache hits from DC miss. This metric is in per thousand instructions (PTI)
	L2 Hit from HWPF	L2 cache hits from L2 hardware prefetching. This metric is in per thousand instructions (PTI)

tlb	L1 ITLB Miss	The instruction fetches that misses in the L1 Instruction Translation Lookaside Buffer(ITLB) but hit in the L2-ITLB plus the ITLB reloads originating from page table walker. The table walk requests are made for L1-ITLB miss and L2-ITLB misses. This metric is in Per-Thousand-Instructions (PTI)
	L2 ITLB Miss	Number of ITLB reloads from page table walker due to L1-ITLB and L2-ITLB misses. This metric is in Per-Thousand-Instructions (PTI)
	L1 DTLB Miss	The number of L1 Data Translation Lookaside Buffer (DTLB) misses from load store micro-ops. This event counts both L2-DTLB hit and L2-DTLB miss. This metric is in Per-Thousand-Instructions (PTI)
	L2 DTLB Miss	The number of L2 Data Translation Lookaside Buffer (DTLB) missed from load store micro-ops. This metric is in Per-Thousand-Instructions (PTI)
	All TLBs Flushed	All TLBs flushed. This metric is in Per-Thousand-Instructions (PTI).
dc	DC Fills from Same CCX	The number of Data Cache (DC) fills from local L2 cache to the core or different L2 cache in the same CCX or L3 cache that belongs to the CCX. This metric is in Per-Thousand-Instructions (PTI)
	DC Fills from different CCX in same node	The number of Data Cache (DC) fills from cache of different CCX in the same package (node). This metric is in Per-Thousand-Instructions (PTI)
	DC Fills from Local Memory	The number of Data Cache (DC) fills from DRAM or IO connected in the same package (node). This metric is in Per-Thousand-Instructions (PTI)

	DC Fills from Remote CCX Cache	The number of Data Cache (DC) fills from cache of CCX in the different package (node). This metric is in Per-Thousand-Instructions (PTI)
	DC Fills from Remote Memory	The number of Data Cache (DC) fills from DRAM or IO connected in the different package (node). This metric is in Per-Thousand-Instructions (PTI)
	All DC Fills	The total number of Data Cache fills from all the data sources. This metric is in Per-Thousand-Instructions (PTI)
l3	L3 Access	L3 cache accesses. This metric is in per thousand instructions (PTI)
	L3 Miss	L3 cache miss. This metric is in per thousand instructions (PTI)
	Ave L3 Miss Latency	Average L3 miss latency in core cycles.
Memory	Mem Ch-A RdBw (GB/s) Mem Ch-A WrBw (GB/s) …	Memory Read and Write bandwidth in GB/s for all the memory channels.
xgmi	xGMI0 BW (GB/s) xGMI1 BW (GB/s) xGMI2 BW (GB/s) xGMI3 BW (GB/s)	Approximate xGMI outbound data bytes in GB/s for all the remote links.

Examples (Linux & FreeBSD)

- Collect IPC data from core 0 for the duration of 60 seconds:

```
# ./AMDuProfPcm -m ipc -c core=0 -d 60 -o /tmp/pcmdata.csv
```

- Collect IPC/L3 metrics for CCX=0 for the duration of 60 seconds:

```
# ./AMDuProfPcm -m ipc,l3 -c ccx=0 -d 60 -o /tmp/pcmdata.csv
```

- Collect only the memory bandwidth across all the UMCs for the duration of 60 seconds and save the output in /tmp/pcmdata.csv file

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

- Collect IPC data for 60 seconds from all the cores:

  ```
  # ./AMDuProfPcm -m ipc -a -d 60 -o /tmp/pcmdata.csv
  ```

- Collect IPC data from core 0 and run the program in core 0:

  ```
  # ./AMDuProfPcm -m ipc -c core=0 -o /tmp/pcmdata.csv -- /usr/bin/taskset -c
  0 myapp.exe
  ```

- Collect IPC and data l2 data from core 0 and report the cumulative (not timeseries) and run the program in core 0

  ```
  # ./AMDuProfPcm -m ipc,l2 -c core=0 -o /tmp/pcmdata.csv -C --
  /usr/bin/taskset -c 0 myapp.exe
  ```

- List the supported raw Core PMC events:

  ```
  # ./AMDuProfPcm -l
  ```

- Print the name, description, and the available unit masks for the specified event:

  ```
  # ./AMDuProfPcm -z pmcx03
  ```

Examples (Windows)

Core Metrics

- To get the list of supported metrics:

  ```
  C:\> AMDuProfPcm.exe -h
  ```

- Collect IPC data from core 0 for the duration of 30 seconds:

  ```
  C:\> AMDuProfPcm.exe -m ipc -c core=0 -d 30 -o c:\tmp\pcmdata.csv
  ```

- Collect IPC/L2 metrics for all the core in CCX=0 for the duration of 30 seconds:

  ```
  C:\> AMDuProfPcm.exe -m ipc,l2 -c ccx=0 -d 30 -o c:\tmp\pcmdata.csv
  ```

- Collect IPC data for 30 seconds from all the cores in the system:

  ```
  C:\> AMDuProfPcm.exe -m ipc -a -d 30 -o c:\tmp\pcmdata.csv
  ```

- Collect IPC data from core 0 and run the program:

  ```
  C:\> AMDuProfPcm.exe -m ipc -c core=0 -o c:\tmp\pcmdata.csv myapp.exe
  ```

- Collect IPC and data l2 data from all the cores and report the aggregated data at the system and package level

[AMD Public Use]

```
C:\> AMDuProfPcm.exe -m ipc,l2 -a -o c:\tmp\pcmdata.csv -d 30 -A
system,package
```

- Collect IPC and data l2 data from all the cores in CCX=0 and report the cumulative (not timeseries)

  ```
  C:\> AMDuProfPcm.exe -m ipc,l2 -c ccx=0 -o c:\tmp\pcmdata.csv -C -d 30
  ```

- Collect IPC and data l2 data from all the cores and report the cumulative (not timeseries)

  ```
  C:\> AMDuProfPcm.exe -m ipc,l2 -a -o c:\tmp\pcmdata.csv -C -d 30
  ```

- Collect IPC and data l2 data from all the cores and report the cumulative (not timeseries) and aggregate at system and package level

  ```
  C:\> AMDuProfPcm.exe -m ipc,l2 -a -o c:\tmp\pcmdata.csv -C -A system,package
  -d 30
  ```

L3 Metrics

- Collect L3 data from ccx=0 for the duration of 30 seconds:

  ```
  C:\> AMDuProfPcm.exe -m l3 -c ccx=0 -d 30 -o c:\tmp\pcmdata.csv
  ```

- Collect L3 data from all the CCXs and report for the duration of 30 seconds:

  ```
  C:\> AMDuProfPcm.exe -m l3 -a -d 30 -o c:\tmp\pcmdata.csv
  ```

- Collect L3 data from all the CCXs and aggregate at system and package level and report for the duration of 30 seconds:

  ```
  C:\> AMDuProfPcm.exe -m l3 -a -d 30 -A system,package -o c:\tmp\pcmdata.csv
  ```

- Collect L3 data from all the CCXs and aggregate at system and package level and report for the duration of 30 seconds: Also report for individual CCXs.

  ```
  C:\> AMDuProfPcm.exe -m l3 -a -d 30 -A system,package,ccx -o
  c:\tmp\pcmdata.csv
  ```

- Collect L3 data from all the CCXs for the duration of 30 seconds and report the cumulative data (no timeseries data)

  ```
  C:\> AMDuProfPcm.exe -m l3 -a -d 30 -C -o c:\tmp\pcmdata.csv
  ```

- Collect L3 data from all the CCXs and aggregate at system and package level and report cumulative data (no timeseries data)

  ```
  C:\> AMDuProfPcm.exe -m l3 -a -d 30 -A system,package -C -o
  c:\tmp\pcmdata.csv
  ```

- Collect IPC data from core 0 for the duration of 30 seconds:

  ```
  C:\> AMDuProfPcm.exe -m ipc -c core=0 -d 30 -o c:\tmp\pcmdata.csv
  ```

Memory Bandwidth:

- Report memory bandwidth for all the memory channels for the duration of 60 seconds and save the output in c:\tmp\pcmdata.csv file

```
C:\> AMDuProfPcm.exe -m memory -a -d 60 -o c:\tmp\pcmdata.csv
```

- Report total memory bandwidth aggregated at the system level for the duration of 60 seconds and save the output in c:\tmp\pcmdata.csv file

```
C:\> AMDuProfPcm.exe -m memory -a -d 60 -o c:\tmp\pcmdata.csv -A system
```

- Report total memory bandwidth aggregated at the system level and also report for every memory channels

```
C:\> AMDuProfPcm.exe -m memory -a -d 60 -o c:\tmp\pcmdata.csv -A
system,package
```

- Report total memory bandwidth aggregated at the system level and also report for all the available memory channels. To report cumulative metric value, instead of timeseries data:

```
C:\> AMDuProfPcm.exe -m memory -a -d 60 -o c:\tmp\pcmdata.csv -C -A
system,package
```

Raw event count dump:

- Monitor events from core 0 and dump the raw event counts for every sample in timeseries manner. No metrics report will be generated

```
C:\> AMDuProfPcm.exe -m ipc -d 60 -D c:\tmp\pcmdata_dump.csv
```

- Monitor events from all the cores and dump the raw event counts for every sample in timeseries manner. No metrics report will be generated

```
C:\> AMDuProfPcm.exe -m ipc -a -d 60 -D c:\tmp\pcmdata_dump.csv
```

Custom config file:

- A sample config XM file is available at <uprof-install-dir>\bin\Data\Config\SamplePcm-core.conf. This file can be copied and modified to specific user-specific interesting events and formula to compute metrics. All the metrics defined in that file, will be monitored, and reported.

```
C:\> AMDuProfPcm.exe -i SamplePcm-core.conf -a -d 60 -o c:\tmp\pcmdata.csv

C:\> AMDuProfPcm.exe -i SamplePcm-core-l3-df.conf -a -d 60 -o
c:\tmp\pcmdata.csv
```

Miscellaneous:

- List the supported raw Core PMC events:

```
C:\> AMDuProfPcm.exe -l
```

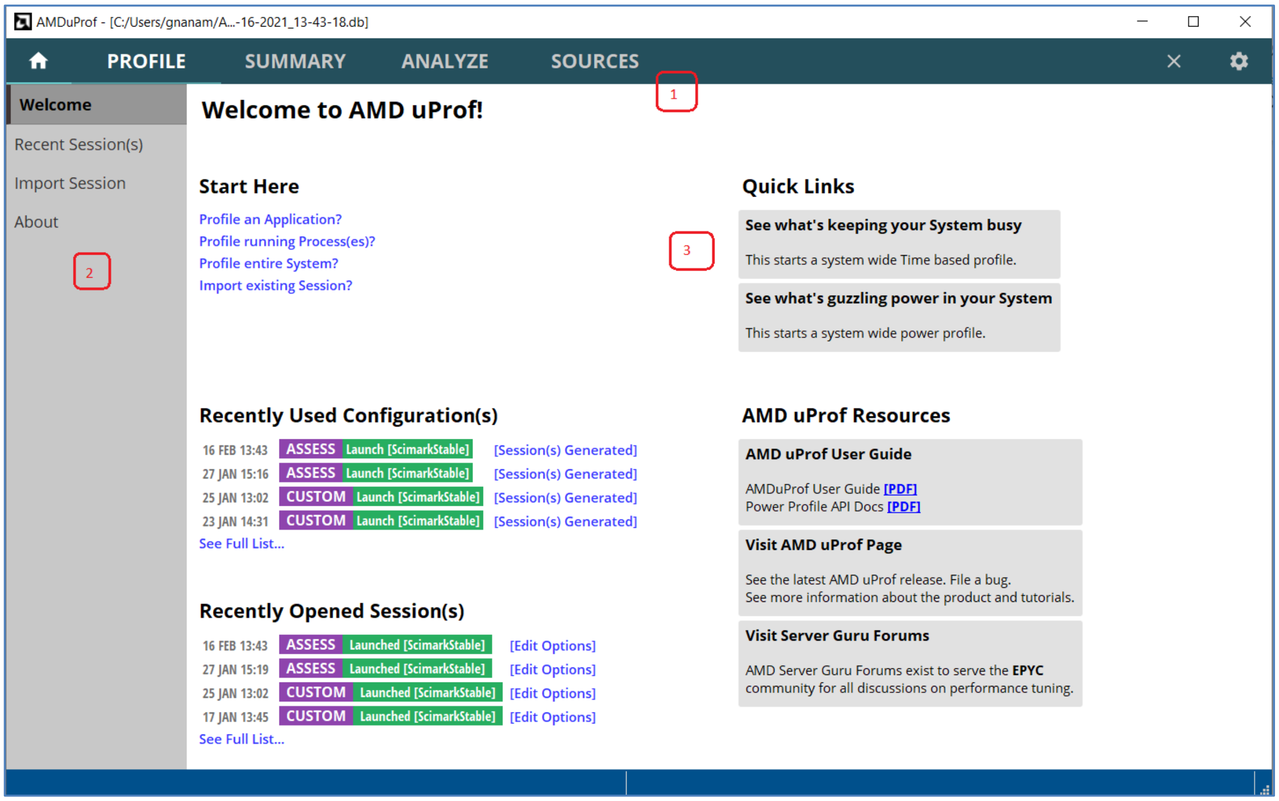
- Print the name, description, and the available unit masks for the specified event:

```
C:\> AMDuProfPcm.exe -z pmcx03
```

Chapter 4 Getting started with AMDuProf GUI

4.1 User Interface

AMDuProf GUI provides a visual interface to profile and analyze the performance data. It has various pages, and each page has several 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.



Welcome to AMD uProf!

Start Here

Profile an Application?
Profile running Process(es)?
Profile entire System?
Import existing Session?

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 Used Configuration(s)

16 FEB 13:43	ASSESS	Launch [ScimarkStable]	[Session(s) Generated]
27 JAN 15:16	ASSESS	Launch [ScimarkStable]	[Session(s) Generated]
25 JAN 13:02	CUSTOM	Launch [ScimarkStable]	[Session(s) Generated]
23 JAN 14:31	CUSTOM	Launch [ScimarkStable]	[Session(s) Generated]

See Full List...

Recently Opened Session(s)

16 FEB 13:43	ASSESS	Launched [ScimarkStable]	[Edit Options]
27 JAN 15:19	ASSESS	Launched [ScimarkStable]	[Edit Options]
25 JAN 13:02	CUSTOM	Launched [ScimarkStable]	[Edit Options]
17 JAN 13:45	CUSTOM	Launched [ScimarkStable]	[Edit Options]

See Full List...

AMD uProf Resources

AMD uProf User Guide
AMDuProf User Guide [PDF]
Power Profile API Docs [PDF]

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

Visit Server Guru Forums
AMD Server Guru Forums exist to serve the **EPYC** community for all discussions on performance tuning.

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 Session(s)**, **Import Session** 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.

4.2 Launching GUI

To launch the AMDuProf GUI program:

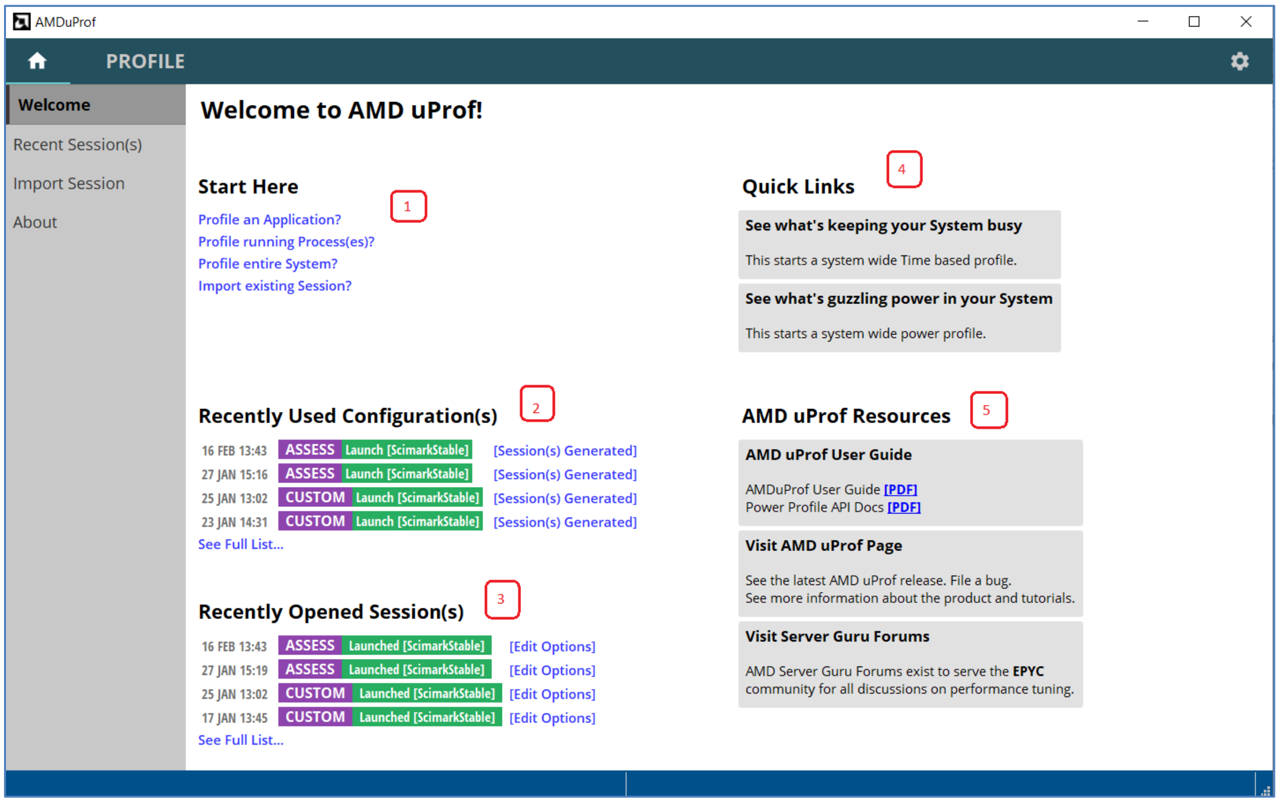
Windows

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

Profile an Application?
Profile running Process(es)?
Profile entire System?
Import existing Session?

Recently Used Configuration(s)

16 FEB 13:43 ASSESS Launch [ScimarkStable] [Session(s) Generated]
27 JAN 15:16 ASSESS Launch [ScimarkStable] [Session(s) Generated]
25 JAN 13:02 CUSTOM Launch [ScimarkStable] [Session(s) Generated]
23 JAN 14:31 CUSTOM Launch [ScimarkStable] [Session(s) Generated]
See Full List...

Recently Opened Session(s)

16 FEB 13:43 ASSESS Launched [ScimarkStable] [Edit Options]
27 JAN 15:19 ASSESS Launched [ScimarkStable] [Edit Options]
25 JAN 13:02 CUSTOM Launched [ScimarkStable] [Edit Options]
17 JAN 13:45 CUSTOM Launched [ScimarkStable] [Edit Options]
See Full List...

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.

AMD uProf Resources

AMD uProf User Guide
AMDuProf User Guide [PDF]
Power Profile API Docs [PDF]

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

Visit Server Guru Forums
AMD Server Guru Forums exist to serve the **EPYC** community for all discussions on performance tuning.

AMDuProf Welcome window

1. **Start Here** section provides quick links to start profile for the various profile targets.
2. Recently used profile configurations are listed in **Recently Used Configuration(s)** section. User can click on this configuration to reuse that profile configuration for subsequent profiling.
3. Recently opened profile sessions are listed in **Recently Opened Session(s)** section. User can click on any one of the sessions to load the corresponding profile data for further analysis.
4. **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.
5. **AMD uProf Resources** section provides links to uProf user guide and power profiler API guide and AMD server community forum for discussions on profiling and performance tuning.

4.3 Configure a profile

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

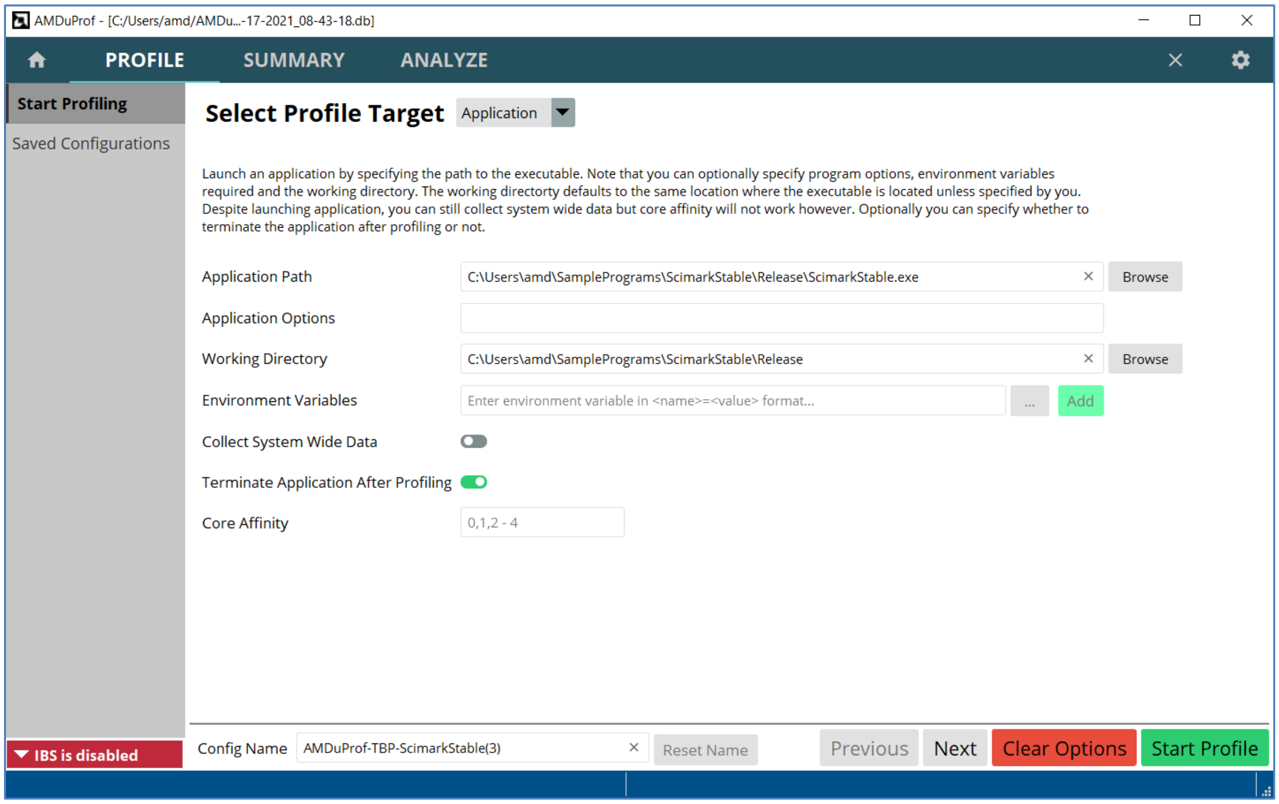
1. Profile target
2. Profile type
    a. What profile data should be collected (CPU or Power performance data)
    b. Monitoring events - how the data should be collected
    c. Additional profile data (if needed) - callstack samples, profile scheduling etc.,

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.

4.3.1 Select Profile Target

To start a profile, either click the **PROFILE** page at the top navigation bar or **Profile an Application?** 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.

Different types of profile target can be selected from the **Select Profile Target** dropdown.



Select Profile Target: Application

Launch an application by specifying the path to the executable. Note that you can optionally specify program options, environment variables required and the 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:\Users\amd\SamplePrograms\ScimarkStable\Release\ScimarkStable.exe
Application Options	
Working Directory	C:\Users\amd\SamplePrograms\ScimarkStable\Release
Environment Variables	Enter environment variable in <name>=<value> format...
Collect System Wide Data	
Terminate Application After Profiling	
Core Affinity	0,1,2 - 4

Config Name: AMDuProf-TBP-ScimarkStable(3)

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).

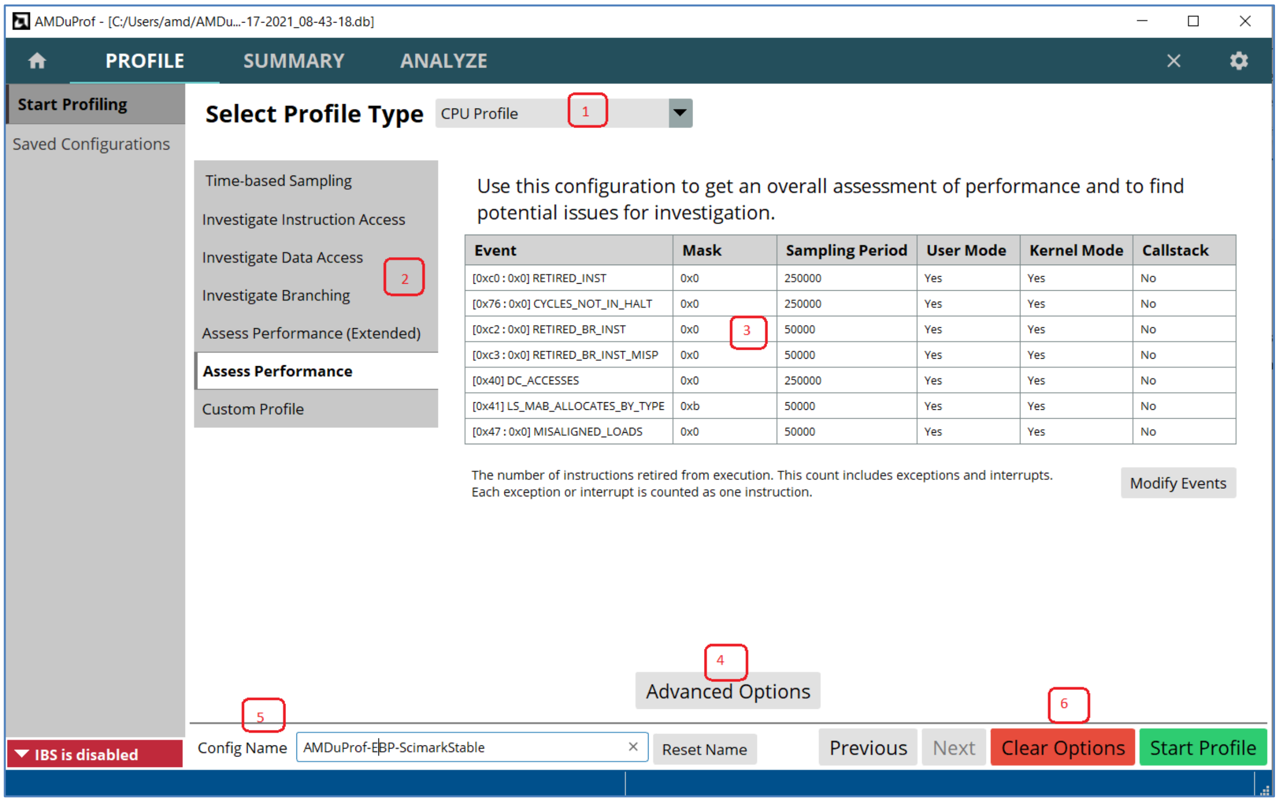
System: Select this if you do not wish to launch any application but perform either a system-wide profile or profile specific set of cores.

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 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.

4.3.2 Select Profile Type

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



Use this configuration to get an overall assessment of performance and to find potential issues for investigation.

Event	Mask	Sampling Period	User Mode	Kernel Mode	Callstack
[0xc0 : 0x0] RETIRED_INST	0x0	250000	Yes	Yes	No
[0x76 : 0x0] CYCLES_NOT_IN_HALT	0x0	250000	Yes	Yes	No
[0xc2 : 0x0] RETIRED_BR_INST	0x0	50000	Yes	Yes	No
[0xc3 : 0x0] RETIRED_BR_INST_MISP	0x0	50000	Yes	Yes	No
[0x40] DC_ACCESSES	0x0	250000	Yes	Yes	No
[0x41] LS_MAB_ALLOCATES_BY_TYPE	0xb	50000	Yes	Yes	No
[0x47 : 0x0] MISALIGNED_LOADS	0x0	50000	Yes	Yes	No

The number of instructions retired from execution. This count includes exceptions and interrupts. Each exception or interrupt is counted as one instruction.

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**, **Sources, and Symbols** etc.,
5. This *profile configuration* details are persistent and saved by the tool with a name – here it is AMDuProf-EBP-ScimarkStable. 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.

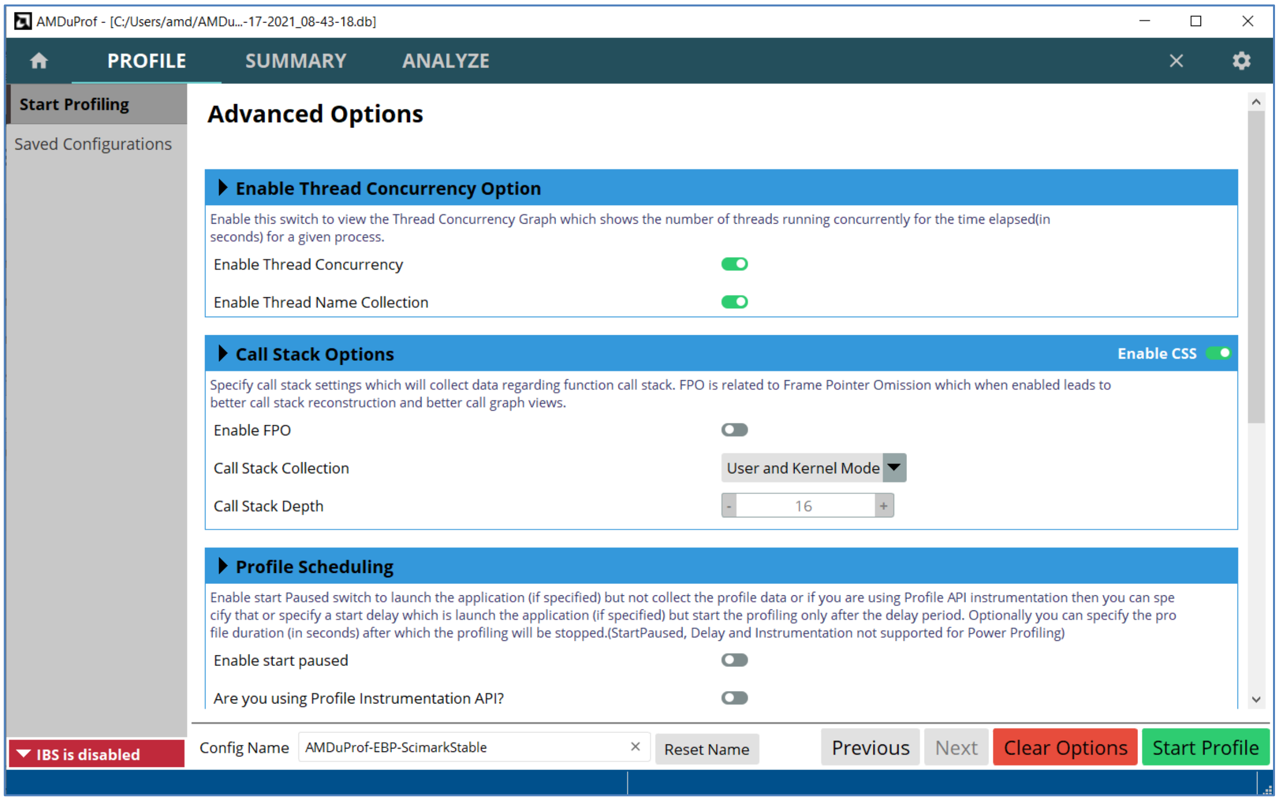
6. The Next and Previous buttons are available to navigate to various fragments within the **Start Profiling** window.

4.3.3 Advanced Options

Advanced options

Clicking **Advanced Options** button in **Select Profile Type** fragment will take you to the **Advanced Options** fragment to set the following options.

1. **Enable Thread Concurrency Option** to collect the profile data to show **Thread Concurrency Chart**. (Windows only option)
2. **Call Stack Options** to enable callstack sample data collection. This profile data is used to show **Flame Graph** and **Call Graph** views.
3. **Profile Scheduling** to schedule the profile data collection.
4. The Next and Previous buttons are available to navigate to various fragments within the **Start Profiling** window.
5. **Sources** line-edit to specify the path(s) to locate the source files of the profiled application.
6. **Symbols** to specify the Symbols servers (Windows only) and to specify the path(s) to locate the symbol files of the profiled application.



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: on
Enable Thread Name Collection: on

Call Stack Options — Enable CSS: on

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: off
Call Stack Collection: User and Kernel Mode
Call Stack Depth: 16

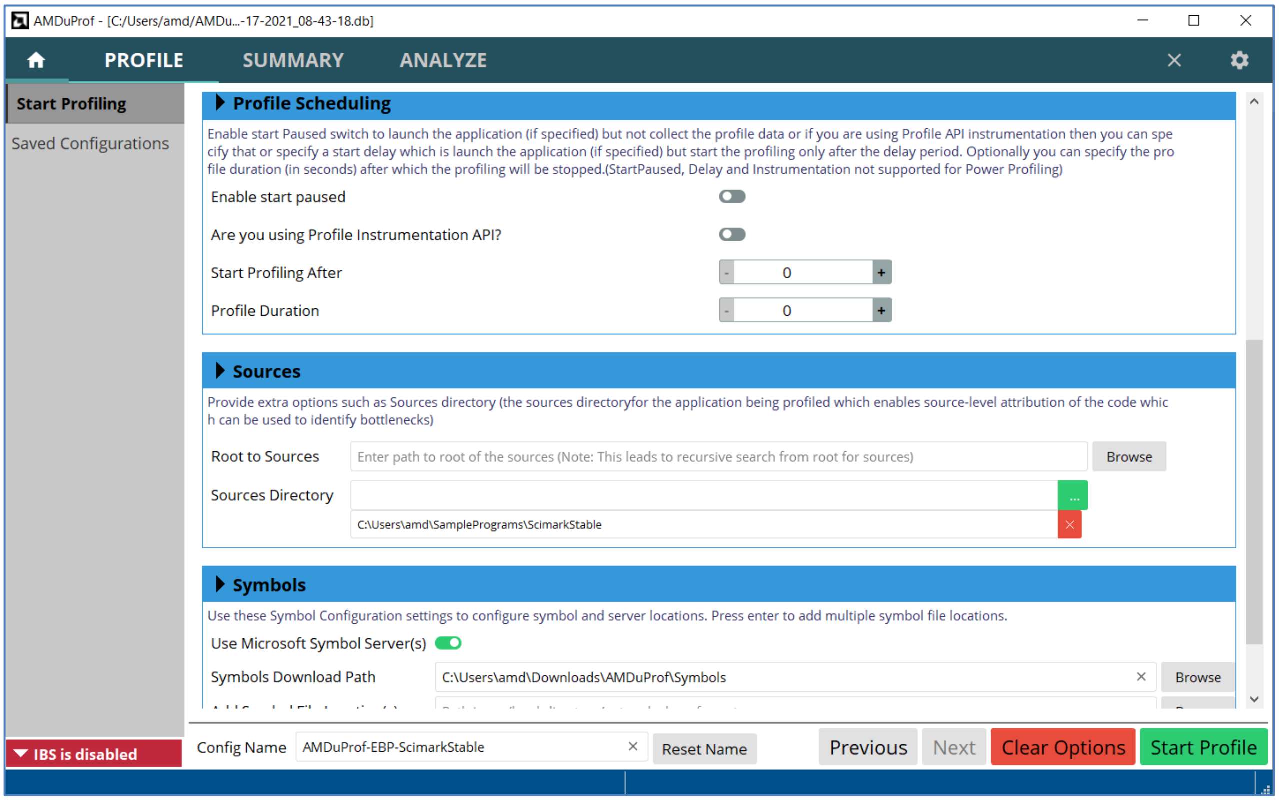
Profile Scheduling

Enable start Paused switch to launch the application (if specified) but not collect the profile data or if you are using Profile API instrumentation then you can specify that or specify a start delay which is launch the application (if specified) but start the profiling only after the delay period. Optionally you can specify the profile duration (in seconds) after which the profiling will be stopped.(StartPaused, Delay and Instrumentation not supported for Power Profiling)

Enable start paused: off
Are you using Profile Instrumentation API?: off

Config Name: AMDuProf-EBP-ScimarkStable
Reset Name | Previous | Next | Clear Options | Start Profile

Start Profiling – Advanced Options



Profile Scheduling

Enable start Paused switch to launch the application (if specified) but not collect the profile data or if you are using Profile API instrumentation then you can specify that or specify a start delay which is launch the application (if specified) but start the profiling only after the delay period. Optionally you can specify the profile duration (in seconds) after which the profiling will be stopped.(StartPaused, Delay and Instrumentation not supported for Power Profiling)

Enable start paused: off
Are you using Profile Instrumentation API?: off
Start Profiling After: 0
Profile Duration: 0

Sources

Provide extra options such as Sources directory (the sources directoryfor the application being profiled which enables source-level attribution of the code which can be used to identify bottlenecks)

Root to Sources: Enter path to root of the sources (Note: This leads to recursive search from root for sources) — Browse
Sources Directory: C:\Users\amd\SamplePrograms\ScimarkStable

Symbols

Use these Symbol Configuration settings to configure symbol and server locations. Press enter to add multiple symbol file locations.

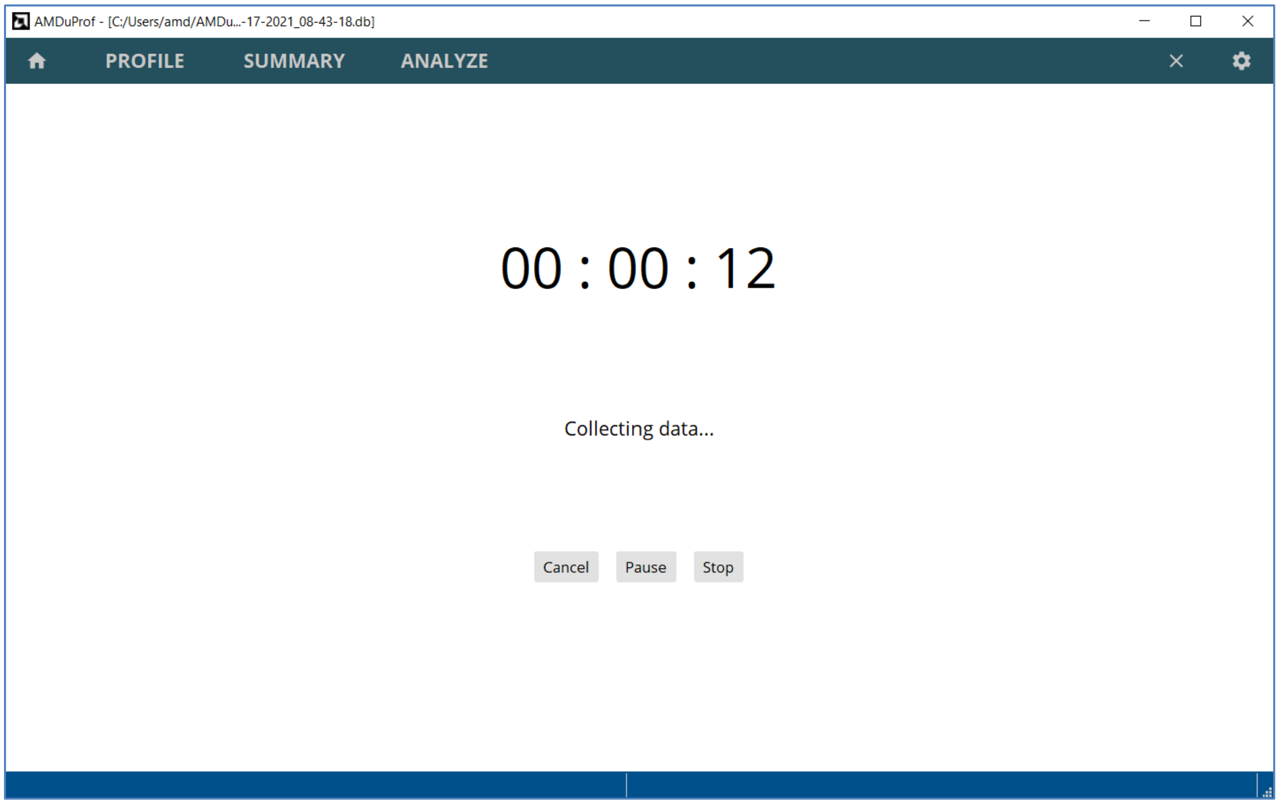
Use Microsoft Symbol Server(s): on
Symbols Download Path: C:\Users\amd\Downloads\AMDuProf\Symbols — Browse

Config Name: AMDuProf-EBP-ScimarkStable
Reset Name | Previous | Next | Clear Options | Start Profile

Start Profiling – Advanced Options

4.3.4 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 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.

4.4 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:

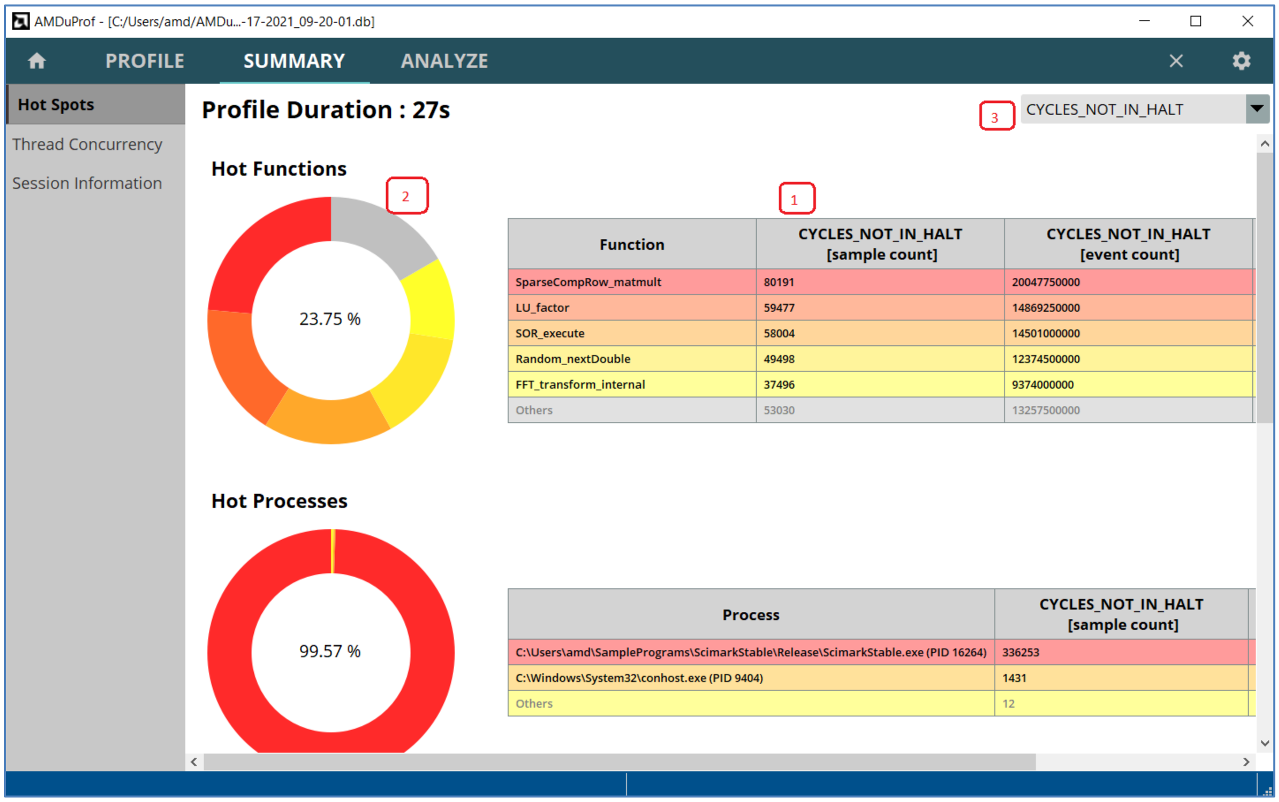
- **SUMMARY** page to look at overview of the hotspots for the profile session.
- **ANALYZE** page to examine the profile data at various granularities.
- **SOURCES** page to examine the data at source line and assembly level.
- **MEMORY** page to examine the cache-line data for potential false cache sharing.
- **HPC** page to examine the OpenMP tracing data for potential load imbalance issue.

The sections available depends on the profile type. The **CPU Profile** and **Power Application Analysis** types will have **SUMMARY**, **ANALYZE, MEMORY, HPC** and **SOURCES** pages to analyze the data.

4.4.1 Overview of performance hotspots

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.



Function	CYCLES_NOT_IN_HALT [sample count]	CYCLES_NOT_IN_HALT [event count]
SparseCompRow_matmult	80191	20047750000
LU_factor	59477	14869250000
SOR_execute	58004	14501000000
Random_nextDouble	49498	12374500000
FFT_transform_internal	37496	9374000000
Others	53030	13257500000

Process	CYCLES_NOT_IN_HALT [sample count]
C:\Users\amd\SamplePrograms\ScimarkStable\Release\ScimarkStable.exe (PID 16264)	336253
C:\Windows\System32\conhost.exe (PID 9404)	1431
Others	12

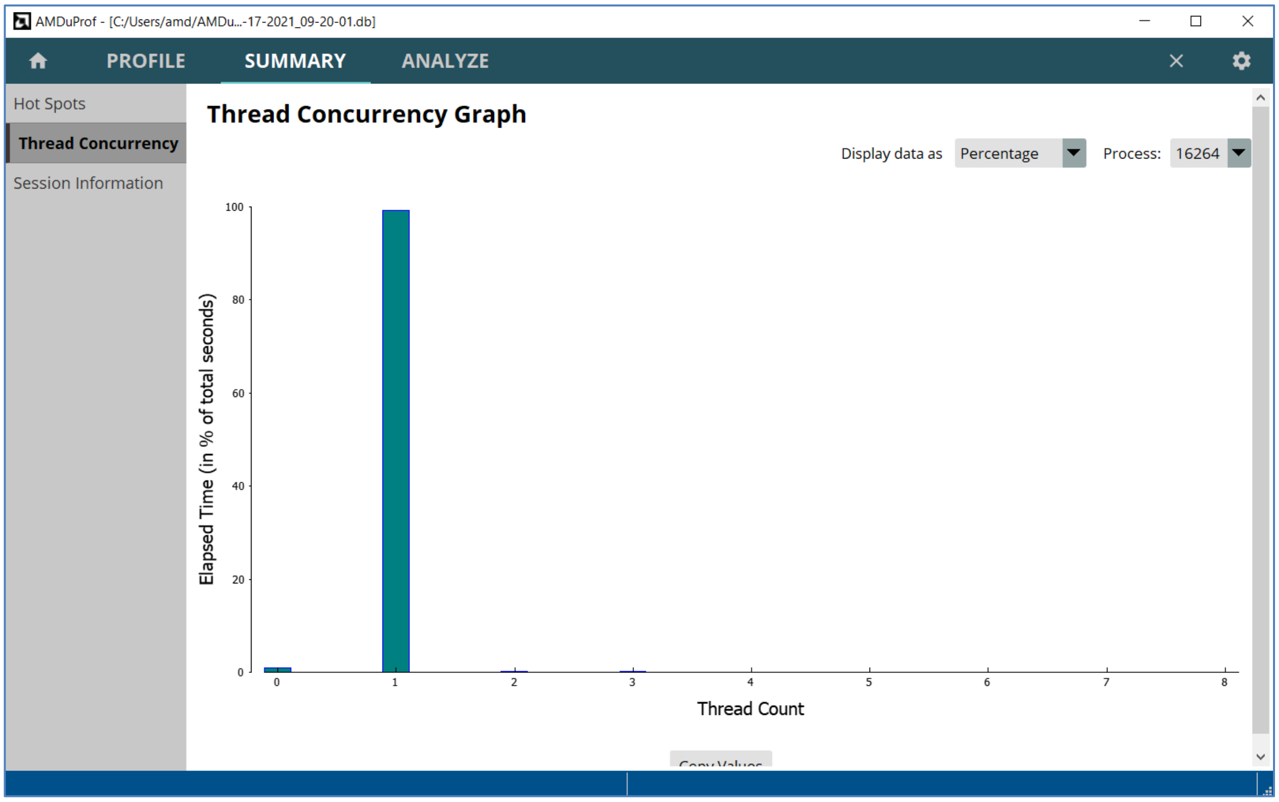
SUMMARY – Hot Spots window

In the above **Hot Spots** window:

1. Lists the top 5 hottest functions, Processes, Modules and Threads for the selected event.
2. The **Hot Functions** pie chart 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 the monitored event can be selected from dropdown in top right corner. Changing it to any other event will update the hotspot data accordingly.

4.4.2 Thread Concurrency Graph

Clicking **SUMMARY → Thread Concurrency** will show the below graph to analyze the thread concurrency of the profiled application. Note: This is Windows OS only feature.



SUMMARY – Thread Concurrency Graph

4.4.3 Function Hotspots

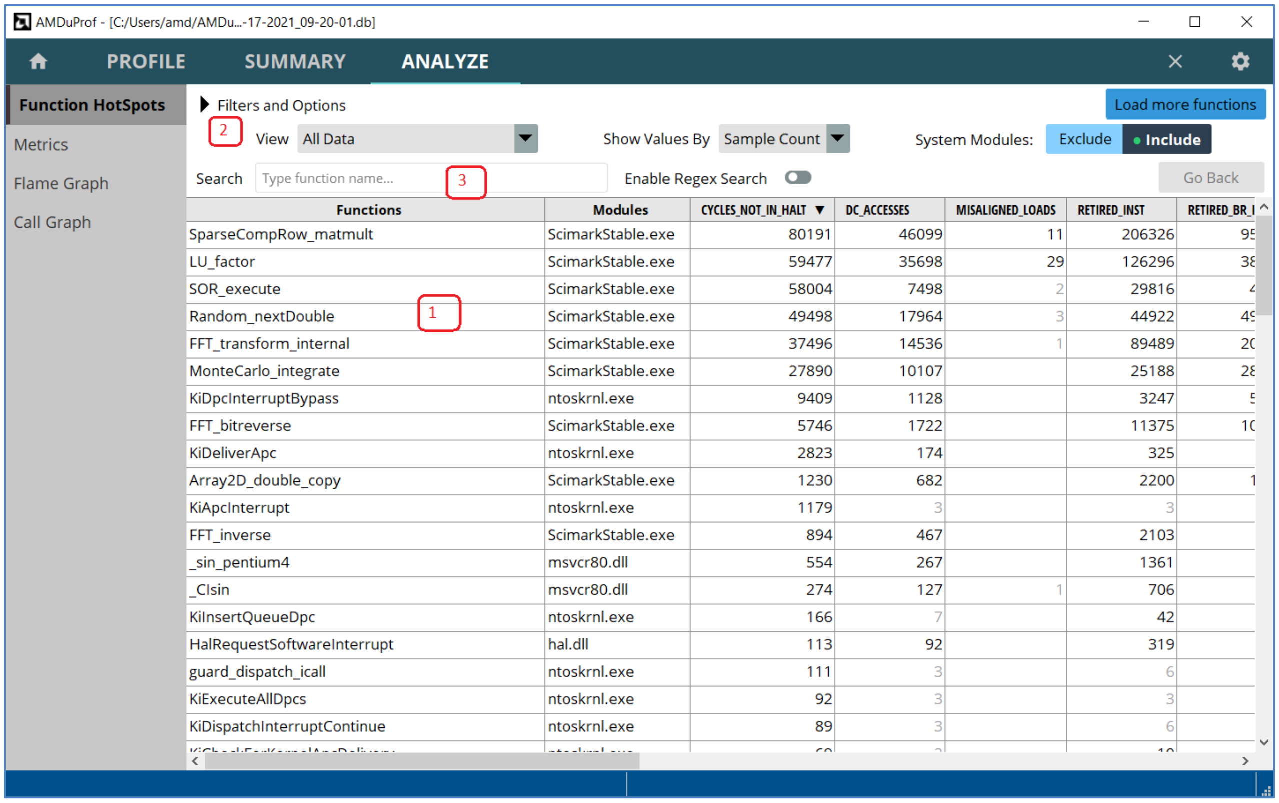
Click on the **ANALYZE** button on the top horizontal navigation bar to go **Function HotSpots** window, which displays the hot functions across all the profiled processes and load modules. This window contains the following:

1. The **Functions** table lists the hot functions - the IP samples are aggregated and attributed at the function-level granularity.
    a) Double click on a function entry to navigate to the corresponding SOURCE view of that function.
    b) Right click will list the following context menu-items
        - “Copy selected rows(s)” to copy the highlighted row to clipboard.
        - “Copy all rows” to copy all the rows to clipboard.

2. **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 **Show Values By** dropdown can be used to display the counter values either as

- "Sample Count" is the number of samples attributed to a function.
- "Event Count" is the product of sample count and sampling interval.
- "Percentage" is the percentage of samples collected for a function.

- The **System Modules** option can be used to either exclude or include the profile data attributed to system modules.



View: All Data
Show Values By: Sample Count
System Modules: Exclude [✓ Include]
Search: Type function name...
Enable Regex Search

Functions	Modules	CYCLES_NOT_IN_HALT ▼	DC_ACCESSES	MISALIGNED_LOADS	RETIRED_INST	RETIRED_BR_I
SparseCompRow_matmult	ScimarkStable.exe	80191	46099	11	206326	95
LU_factor	ScimarkStable.exe	59477	35698	29	126296	38
SOR_execute	ScimarkStable.exe	58004	7498	2	29816	4
Random_nextDouble	ScimarkStable.exe	49498	17964	3	44922	49
FFT_transform_internal	ScimarkStable.exe	37496	14536	1	89489	20
MonteCarlo_integrate	ScimarkStable.exe	27890	10107		25188	28
KiDpcInterruptBypass	ntoskrnl.exe	9409	1128		3247	5
FFT_bitreverse	ScimarkStable.exe	5746	1722		11375	10
KiDeliverApc	ntoskrnl.exe	2823	174		325	
Array2D_double_copy	ScimarkStable.exe	1230	682		2200	1
KiApcInterrupt	ntoskrnl.exe	1179	3		3	
FFT_inverse	ScimarkStable.exe	894	467		2103	
_sin_pentium4	msvcr80.dll	554	267		1361	
_CIsin	msvcr80.dll	274	127	1	706	
KiInsertQueueDpc	ntoskrnl.exe	166	7		42	
HalRequestSoftwareInterrupt	hal.dll	113	92		319	
guard_dispatch_icall	ntoskrnl.exe	111	3		6	
KiExecuteAllDpcs	ntoskrnl.exe	92	3		3	
KiDispatchInterruptContinue	ntoskrnl.exe	89	3		6	

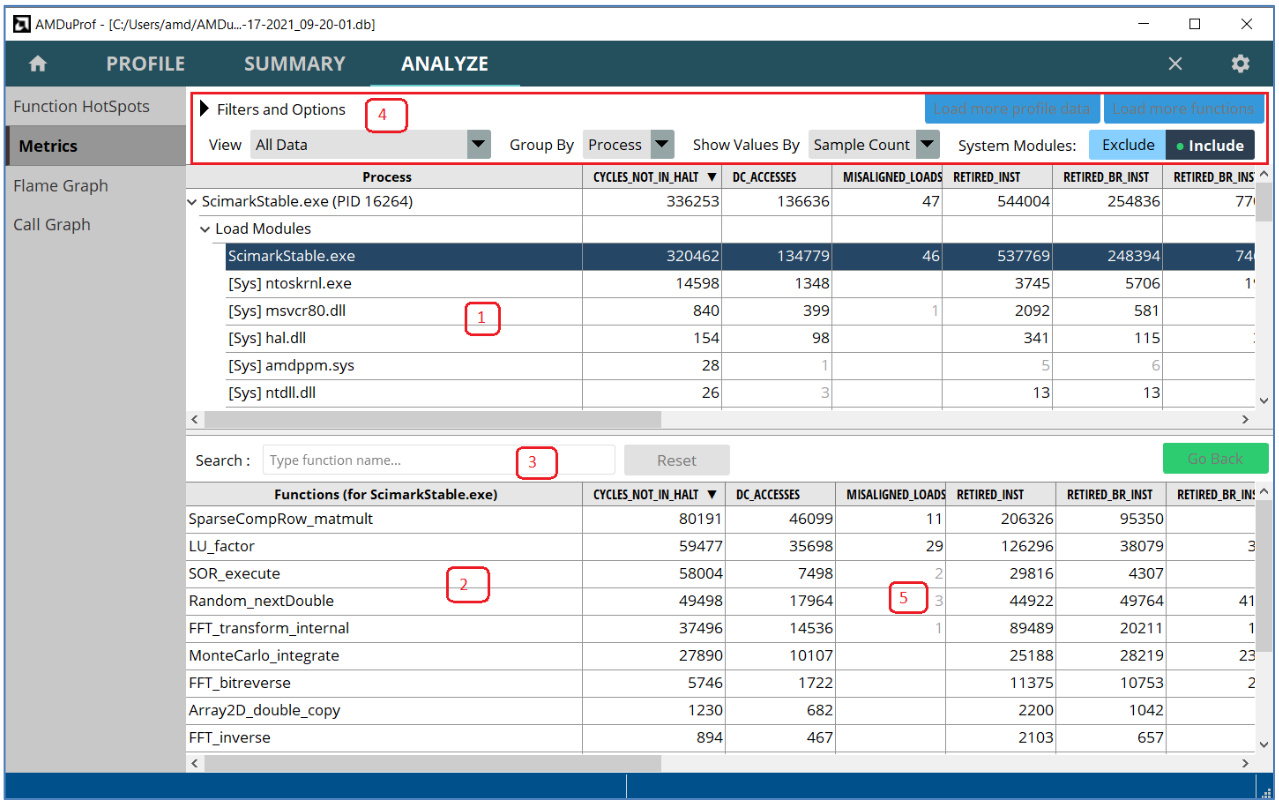
ANALYZE – Function HotSpots

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.
   a) Click **Go Back** button to go back to the **Functions** table that list all the functions.
   b) Turn on **Enable Regex Search** switch to search with regular expression matching.

Not all entries will be loaded for a profile. To load more than the default number of entries, click the **Load more functions** button on the top right corner to display more data. The columns can be sorted as well by clicking on the column headers.

4.4.4 Process and Functions

Clicking **ANALYZE → Metrics** will display the profile data table at various program unit granularities - Process, Load Modules, Threads and Functions. The window contains data in two different formats:



View: All Data Group By: Process Show Values By: Sample Count System Modules: Exclude ● Include

Search : Type function name...

Process	CYCLES_NOT_IN_HALT ▼	DC_ACCESSES	MISALIGNED_LOADS	RETIRED_INST	RETIRED_BR_INST	RETIRED_BR_INS
ScimarkStable.exe (PID 16264)	336253	136636	47	544004	254836	77
Load Modules						
ScimarkStable.exe	320462	134779	46	537769	248394	74
[Sys] ntoskrnl.exe	14598	1348		3745	5706	1
[Sys] msvcr80.dll	840	399	1	2092	581	
[Sys] hal.dll	154	98		341	115	
[Sys] amdppm.sys	28	1		5	6	
[Sys] ntdll.dll	26	3		13	13	

Functions (for ScimarkStable.exe)	CYCLES_NOT_IN_HALT ▼	DC_ACCESSES	MISALIGNED_LOADS	RETIRED_INST	RETIRED_BR_INST	RETIRED_BR_INS
SparseCompRow_matmult	80191	46099	11	206326	95350	
LU_factor	59477	35698	29	126296	38079	3
SOR_execute	58004	7498	2	29816	4307	
Random_nextDouble	49498	17964	3	44922	49764	41
FFT_transform_internal	37496	14536	1	89489	20211	1
MonteCarlo_integrate	27890	10107		25188	28219	23
FFT_bitreverse	5746	1722		11375	10753	2
Array2D_double_copy	1230	682		2200	1042	
FFT_inverse	894	467		2103	657	

ANALYZE page - Metrics window

1. The upper tree represents 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.
    a) Right click will list the following context menu-items
        - "Expand All Entries" to list the modules and threads of all the processes.
        - "Collapse All Entries" to list only the top-level entries.
        - "Copy selected rows(s)" to copy the highlighted row to clipboard.
        - "Copy all rows" to copy all the rows to clipboard.

2. The lower **Functions** table contains samples attributed to corresponding functions. The function entries depend 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.
    a) Double click on a function entry to navigate to the corresponding SOURCE view of that function.
    b) Right click will list the following context menu-items
        - "Copy selected rows(s)" to copy the highlighted row to clipboard.

    - "Copy all rows" to copy all the rows to clipboard.
    - "Open Call Graph" to navigate to the corresponding function entry in **Call Graph** section.

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.
    a) Click **Reset** button to clear the search text box.
    b) Click **Go Back** button to go back to the **Functions** table that list all the functions.

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** dropdown is used to group the data by Process, Module and Thread. By default, the sample data is grouped-by Process
    - The **Show Values By** dropdown can be used to display the counter values either as
        - "Sample Count" is the number of samples attributed to a function.
        - "Event Count" is the product of sample count and sampling interval.
        - "Percentage" is the percentage of samples collected for a function.
    - The **System Modules** option can be used to either exclude or include the profile data attributed to system modules.

5. Confidence level - The metrics that cannot be calculated reliably due to low number of samples collected for a program unit will be greyed out.

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 corner to fetch more data. The columns can be sorted as well by clicking on the column headers.

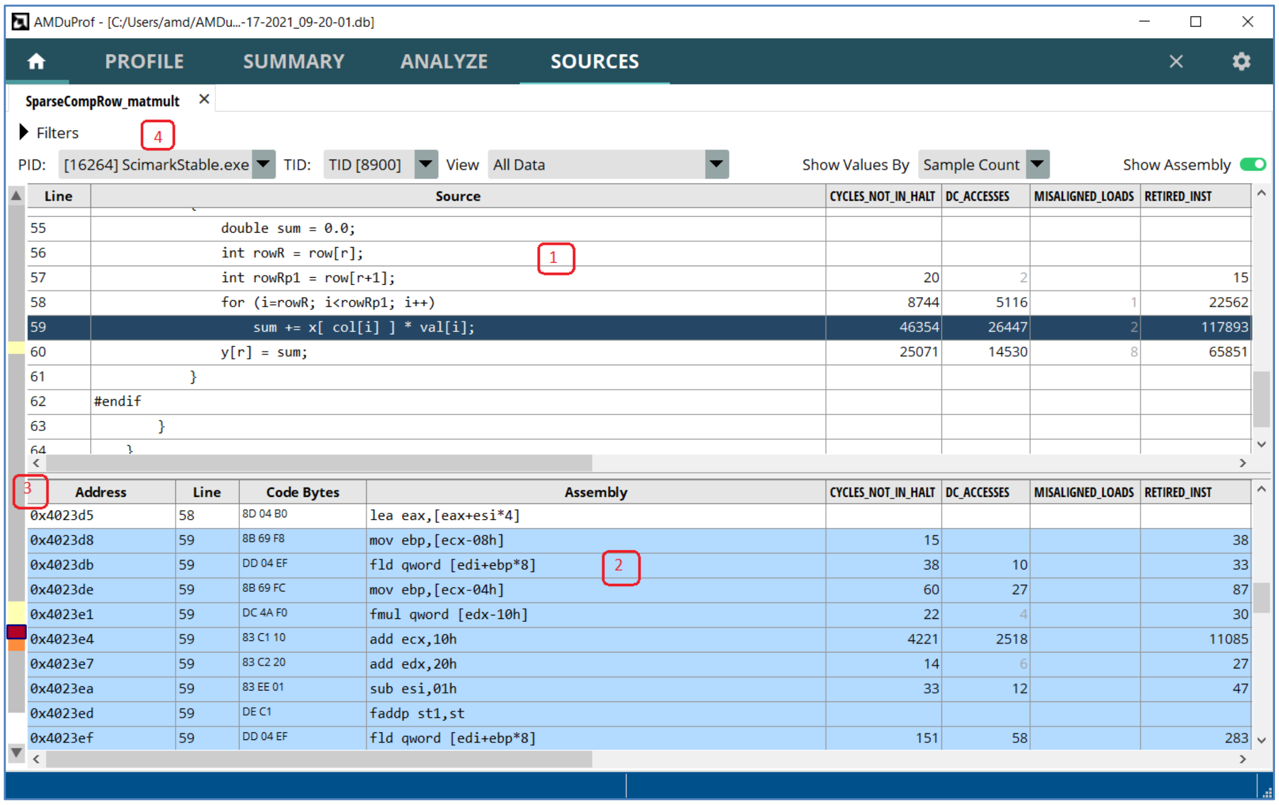
4.4.5 Source and Assembly

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

Following section are there in the source tab:

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. The assembly instruction of the corresponding highlighted source line. The tree will also show the offset for each assembly instruction along with metrics.
3. Heatmap – overview of the hotspots at source level.



PID: [16264] ScimarkStable.exe TID: TID [8900] View All Data Show Values By Sample Count Show Assembly

Line	Source	CYCLES_NOT_IN_HALT	DC_ACCESSES	MISALIGNED_LOADS	RETIRED_INST
55	`double sum = 0.0;`				
56	`int rowR = row[r];`				
57	`int rowRp1 = row[r+1];`	20	2		15
58	`for (i=rowR; i<rowRp1; i++)`	8744	5116	1	22562
59	`sum += x[ col[i] ] * val[i];`	46354	26447	2	117893
60	`y[r] = sum;`	25071	14530	8	65851
61	`}`				
62	`#endif`				
63	`}`				
64	`}`				

Address	Line	Code Bytes	Assembly	CYCLES_NOT_IN_HALT	DC_ACCESSES	MISALIGNED_LOADS	RETIRED_INST
0x4023d5	58	8D 04 B0	lea eax,[eax+esi*4]				
0x4023d8	59	8B 69 F8	mov ebp,[ecx-08h]	15			38
0x4023db	59	DD 04 EF	fld qword [edi+ebp*8]	38	10		33
0x4023de	59	8B 69 FC	mov ebp,[ecx-04h]	60	27		87
0x4023e1	59	DC 4A F0	fmul qword [edx-10h]	22	4		30
0x4023e4	59	83 C1 10	add ecx,10h	4221	2518		11085
0x4023e7	59	83 C2 20	add edx,20h	14	6		27
0x4023ea	59	83 EE 01	sub esi,01h	33	12		47
0x4023ed	59	DE C1	faddp st1,st				
0x4023ef	59	DD 04 EF	fld qword [edi+ebp*8]	151	58		283

SOURCES – source and assembly window

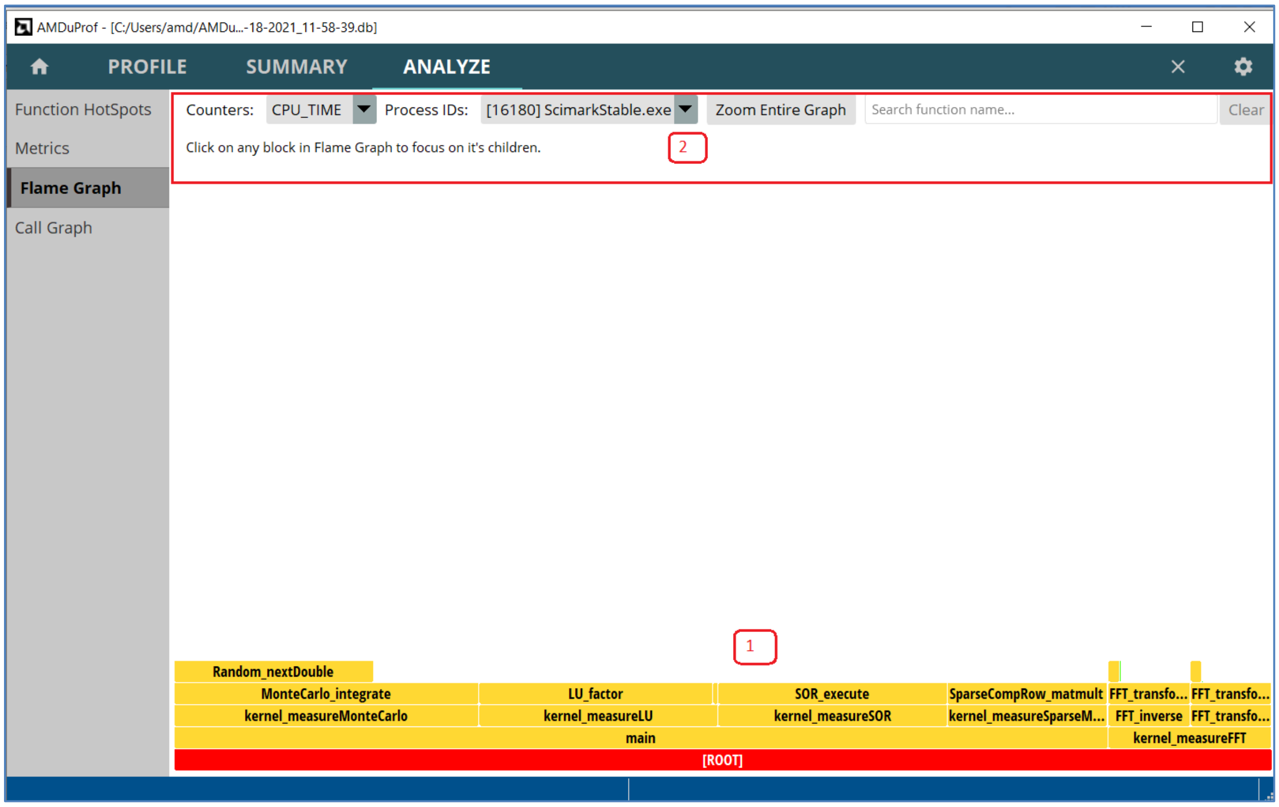
4. **Filters** pane lets you filter the profile data by providing the following options.
   - The **View** controls the counters that are displayed. The relevant counters and their derived metrics are grouped in predefined views. The user can select it from the **View** drop-down. Refer *this* section 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 By** dropdown can be used to display the counter values either as
     - "Sample Count” is the number of samples attributed to a function.
     - “Event Count” is the product of sample count and sampling interval.
     - “Percentage” is the percentage of samples collected for a function.

For multi-threaded or multi-process applications, if a function has been executed from multiple threads or processes, then each of them will be listed in the **PID** and **TID** dropdowns in **Filters** pane. Changing them will update the profile data for that 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.

4.4.6 Flame Graph

Flame graph is a visualization of sampled callstack traces to quickly identify the hottest code execution paths. Clicking **ANALYZE → Flame Graph** will navigate to this window.



Flame graph window

The Flamegraph section has:

1. The x-axis of the Flamegraph shows the callstack profile and the y-axis shows the stack depth. It is not plotted based on passage of time. Each cell represents a stack frame and if a frame were present more often in the callstack samples, the cell would be wider.
    a) Module wise coloring of the cells.
    b) Clicking on a cell will zoom only that cell and its children. Use Reset Zoom button visualize the entire graph.
    c) Right click on a cell will list the following context menu-items

- “Copy Function Data” to copy the function names and its metrics to clipboard.
- “Open Source View” to navigate to the source tab of that function.

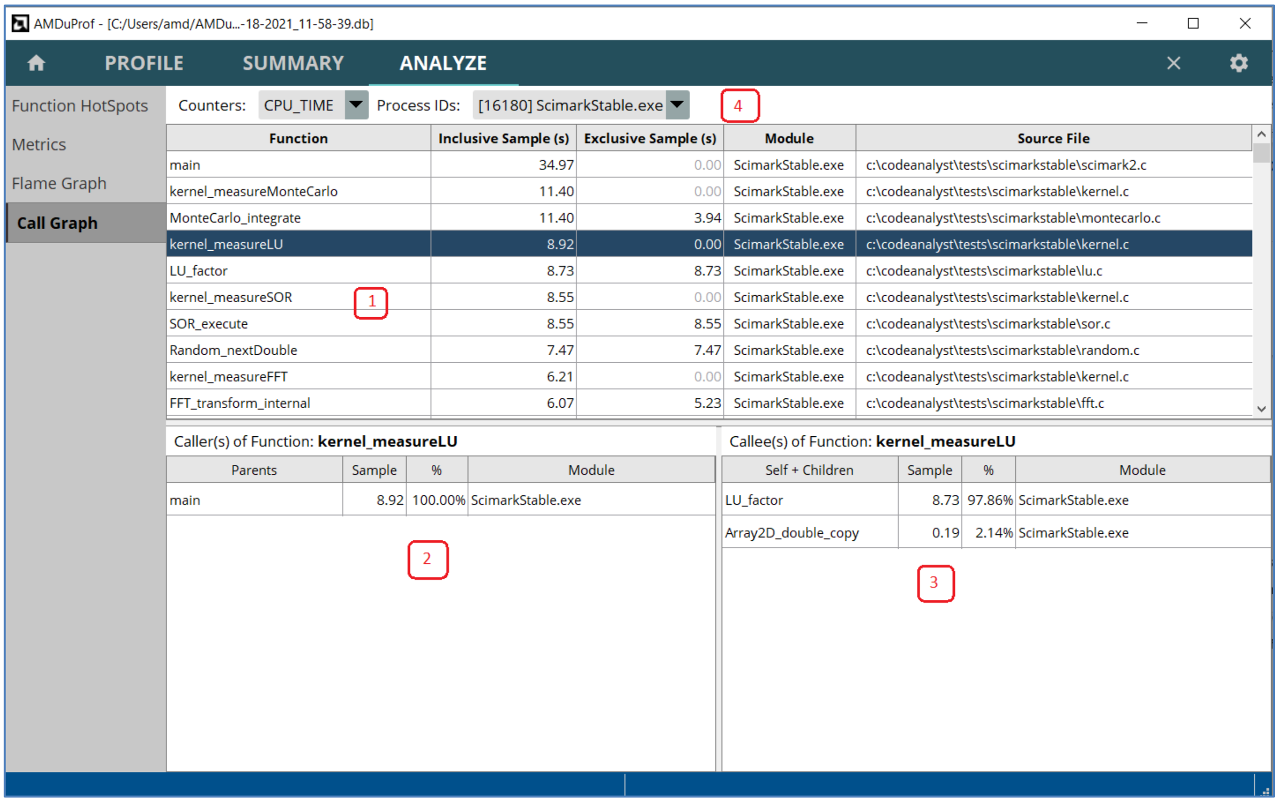
d) Hovering the mouse over a cell will display the tooltip showing the inclusive and exclusive number of samples of that function.

2. Following options are available at the top of this section.

- Click **Zoom Entire Graph** button for better zooming experience.
- Searching for particular function will highlight that function cells in all the Flamegraph.
- The **Process IDs** dropdown lists all the processes for which callstack samples are collected. Changing the process will plot the Flamegraph for that particular process.
- The **Counters** dropdown lists all the counters for which callstack samples are collected. Changing the counter will plot the Flamegraph for that particular counter.

4.4.7 Callgraph

Clicking **ANALYZE → Call Graph** will navigate to the callgraph windows. This is constructed using the callstack samples and offers a butterfly view to analyze the hot call-paths.



Counters: CPU_TIME Process IDs: [16180] ScimarkStable.exe

Function	Inclusive Sample (s)	Exclusive Sample (s)	Module	Source File
main	34.97	0.00	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\scimark2.c
kernel_measureMonteCarlo	11.40	0.00	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\kernel.c
MonteCarlo_integrate	11.40	3.94	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\montecarlo.c
kernel_measureLU	8.92	0.00	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\kernel.c
LU_factor	8.73	8.73	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\lu.c
kernel_measureSOR	8.55	0.00	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\kernel.c
SOR_execute	8.55	8.55	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\sor.c
Random_nextDouble	7.47	7.47	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\random.c
kernel_measureFFT	6.21	0.00	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\kernel.c
FFT_transform_internal	6.07	5.23	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\fft.c

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

Parents	Sample	%	Module
main	8.92	100.00%	ScimarkStable.exe

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

Self + Children	Sample	%	Module
LU_factor	8.73	97.86%	ScimarkStable.exe
Array2D_double_copy	0.19	2.14%	ScimarkStable.exe

ANALYZE – Call graph window

1. The Function table lists all the functions with inclusive and excusive samples.
   a) Double clicking on a function entry having exclusive samples will navigate to the corresponding function source view.
   b) Right click on an entry will list the following context menu-items
      - "Copy Rows(s)" to copy the highlighted row to clipboard.
      - "Copy All Rows" to copy all the rows to clipboard.
   c) Clicking on function will show its Caller and Callee functions in the butterfly view.
2. Lists all the parents of the function selected in the Function table.
   a) Right click on an entry will list the following context menu-items
      - "Go to Caller/Callee" to show the parent and children of this function.
      - "Copy Current Rows" to copy the highlighted row to clipboard.
      - "Copy All Row(s)" to copy all the rows to clipboard.
3. Lists all the children of the function selected in the Function table.
   a) Right click on an entry will list the following context menu-items
      - "Go to Caller/Callee" to show the parent and children of that function.
      - "Copy Current Rows" to copy the highlighted row to clipboard.
      - "Copy All Row(s)" to copy all the rows to clipboard.
4. Options
   - The **Process IDs** dropdown lists all the processes for which callstack samples are collected. Changing the process will show the callgraph for that particular process.
   - The **Counters** dropdown lists all the counters for which callstack samples are collected. Changing the counter will show the callgraph for that particular counter.

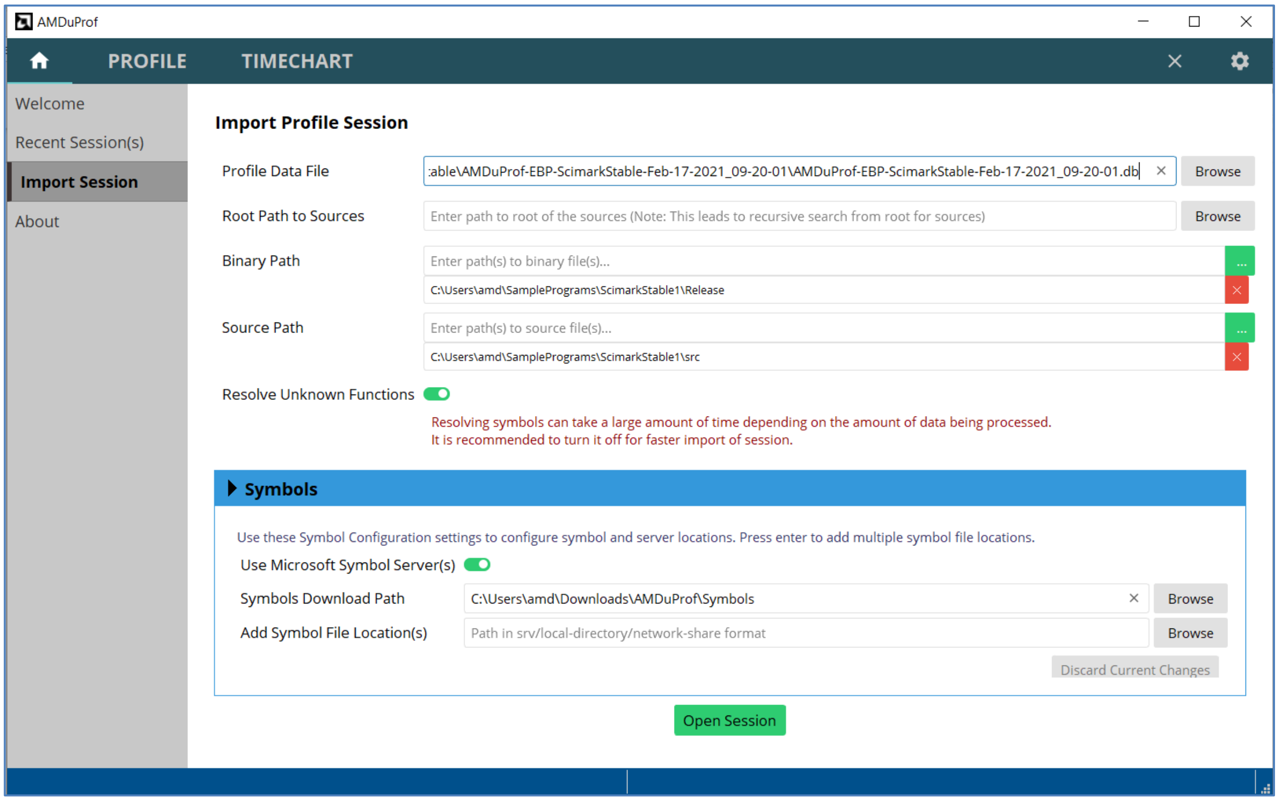
4.5 Importing Profile Databases

To analyze a profile database generated using CLI, clicking **HOME → Import Session** will navigate to **Import Profile Session** window and you will see the following window.

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

- **Symbols**: Specify the symbol path(s) from where the debug info files (On Windows, PDB files) can be located.



Import Profile Session

Profile Data File: :able\AMDuProf-EBP-ScimarkStable-Feb-17-2021_09-20-01\AMDuProf-EBP-ScimarkStable-Feb-17-2021_09-20-01.db

Root Path to Sources: Enter path to root of the sources (Note: This leads to recursive search from root for sources)

Binary Path: Enter path(s) to binary file(s)... C:\Users\amd\SamplePrograms\ScimarkStable1\Release

Source Path: Enter path(s) to source file(s)... C:\Users\amd\SamplePrograms\ScimarkStable1\src

Resolve Unknown Functions

Resolving symbols can take a large amount of time depending on the amount of data being processed.
It is recommended to turn it off for faster import of session.

Symbols

Use these Symbol Configuration settings to configure symbol and server locations. Press enter to add multiple symbol file locations.

Use Microsoft Symbol Server(s)

Symbols Download Path: C:\Users\amd\Downloads\AMDuProf\Symbols

Add Symbol File Location(s): Path in srv/local-directory/network-share format

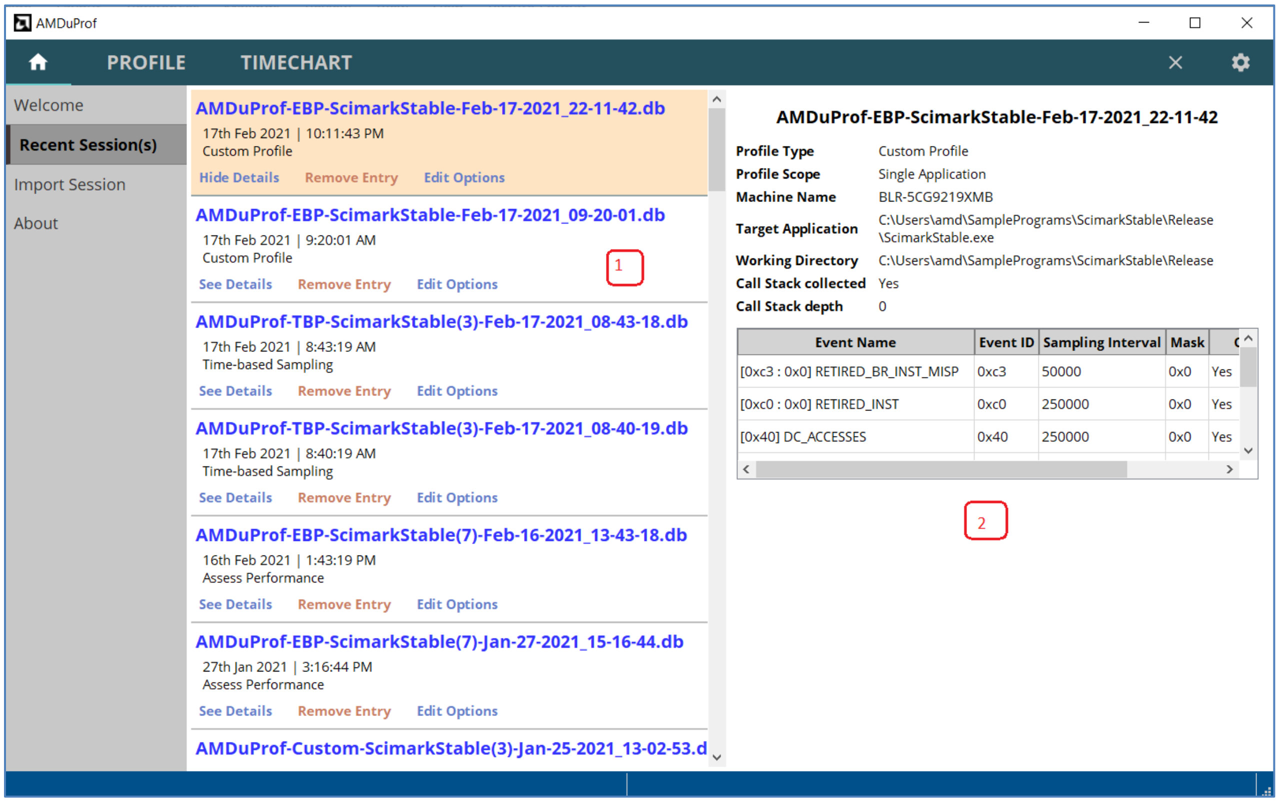
Import Session – importing profile database

4.6 Analyzing saved Profile Session

Once you have a created new profile session or opened(imported) profile database, the history is updated 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 → Recent Session(s)** as well.

In the below screenshot:

1. History of profile sessions opened for analysis in the GUI.
   a) Clicking on an entry will load the corresponding profile db for analysis.
   b) **See Details** button will show details about this profile session like profiled application, monitored events list etc.,
   c) Clicking **Edit Options** will automatically fill the **Import Profile Session** for this db and let you update any of the line-edits before opening the session.
   d) **Remove Entry** button will delete this profile session from the history.
2. Details of the selected profile session.


AMDuProf

AMDuProf-EBP-ScimarkStable-Feb-17-2021_22-11-42.db
17th Feb 2021 | 10:11:43 PM
Custom Profile
Hide Details Remove Entry Edit Options

AMDuProf-EBP-ScimarkStable-Feb-17-2021_09-20-01.db
17th Feb 2021 | 9:20:01 AM
Custom Profile
See Details Remove Entry Edit Options

AMDuProf-TBP-ScimarkStable(3)-Feb-17-2021_08-43-18.db
17th Feb 2021 | 8:43:19 AM
Time-based Sampling
See Details Remove Entry Edit Options

AMDuProf-TBP-ScimarkStable(3)-Feb-17-2021_08-40-19.db
17th Feb 2021 | 8:40:19 AM
Time-based Sampling
See Details Remove Entry Edit Options

AMDuProf-EBP-ScimarkStable(7)-Feb-16-2021_13-43-18.db
16th Feb 2021 | 1:43:19 PM
Assess Performance
See Details Remove Entry Edit Options

AMDuProf-EBP-ScimarkStable(7)-Jan-27-2021_15-16-44.db
27th Jan 2021 | 3:16:44 PM
Assess Performance
See Details Remove Entry Edit Options

AMDuProf-Custom-ScimarkStable(3)-Jan-25-2021_13-02-53.d

AMDuProf-EBP-ScimarkStable-Feb-17-2021_22-11-42

Profile Type	Custom Profile
Profile Scope	Single Application
Machine Name	BLR-5CG9219XMB
Target Application	C:\Users\amd\SamplePrograms\ScimarkStable\Release\ScimarkStable.exe
Working Directory	C:\Users\amd\SamplePrograms\ScimarkStable\Release
Call Stack collected	Yes
Call Stack depth	0

Event Name	Event ID	Sampling Interval	Mask	C
[0xc3 : 0x0] RETIRED_BR_INST_MISP	0xc3	50000	0x0	Yes
[0xc0 : 0x0] RETIRED_INST	0xc0	250000	0x0	Yes
[0x40] DC_ACCESSES	0x40	250000	0x0	Yes

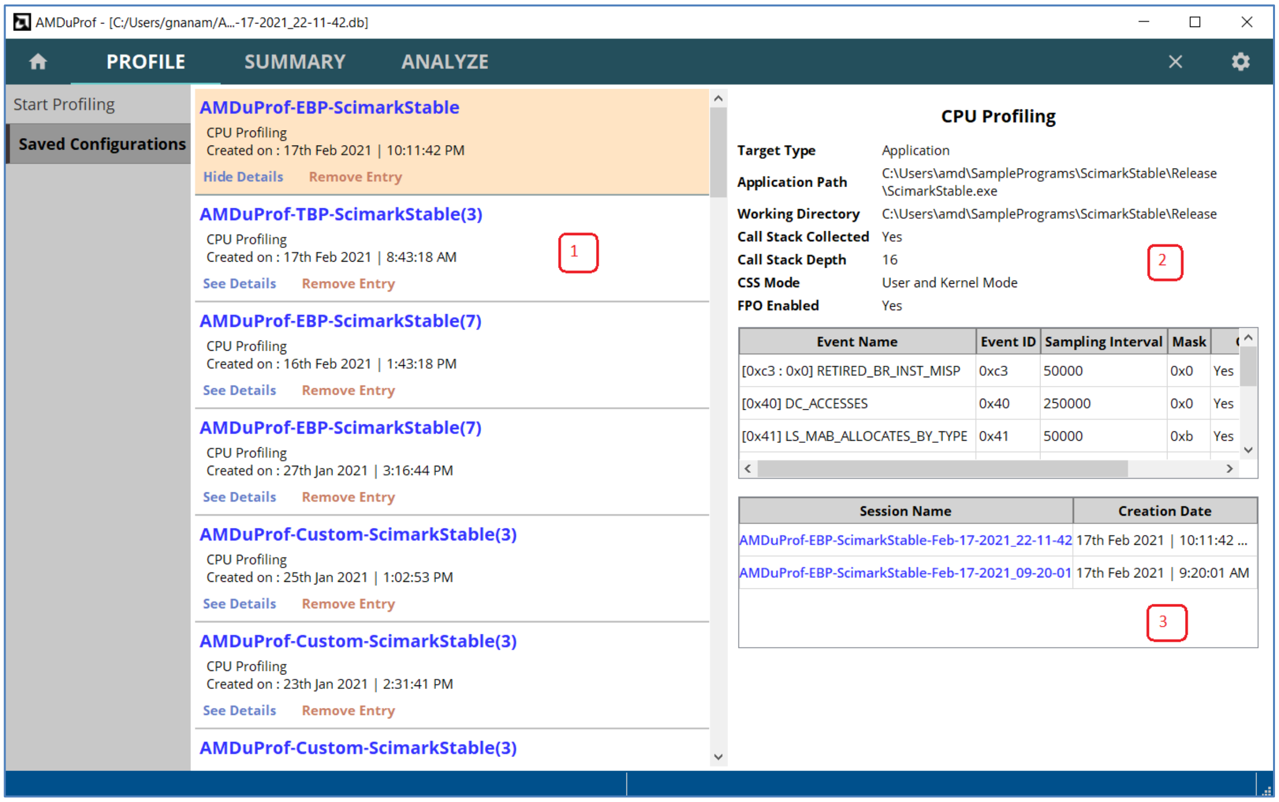
PROFILE – Recent Sessions

4.7 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 → Saved Configurations** window.

In the below screenshot:

1. History of profile configurations used to collect profile data using GUI.
   a) Clicking on an entry will load the corresponding profile configuration for data collection.
   b) **See Details** button will show details about this profile session like profiled application, monitored events list etc.,
   c) **Remove Entry** button will delete this profile session from the history.
2. Details of the selected profile session.
3. History of generated sessions using this profile configuration.
   a) Clicking on an entry will load the profile session db for analysis.



AMDuProf-EBP-ScimarkStable
CPU Profiling
Created on : 17th Feb 2021 | 10:11:42 PM
Hide Details Remove Entry

AMDuProf-TBP-ScimarkStable(3)
CPU Profiling
Created on : 17th Feb 2021 | 8:43:18 AM
See Details Remove Entry

AMDuProf-EBP-ScimarkStable(7)
CPU Profiling
Created on : 16th Feb 2021 | 1:43:18 PM
See Details Remove Entry

AMDuProf-EBP-ScimarkStable(7)
CPU Profiling
Created on : 27th Jan 2021 | 3:16:44 PM
See Details Remove Entry

AMDuProf-Custom-ScimarkStable(3)
CPU Profiling
Created on : 25th Jan 2021 | 1:02:53 PM
See Details Remove Entry

AMDuProf-Custom-ScimarkStable(3)
CPU Profiling
Created on : 23th Jan 2021 | 2:31:41 PM
See Details Remove Entry

AMDuProf-Custom-ScimarkStable(3)

CPU Profiling

Target Type	Application
Application Path	C:\Users\amd\SamplePrograms\ScimarkStable\Release \ScimarkStable.exe
Working Directory	C:\Users\amd\SamplePrograms\ScimarkStable\Release
Call Stack Collected	Yes
Call Stack Depth	16
CSS Mode	User and Kernel Mode
FPO Enabled	Yes

Event Name	Event ID	Sampling Interval	Mask	(
[0xc3 : 0x0] RETIRED_BR_INST_MISP	0xc3	50000	0x0	Yes
[0x40] DC_ACCESSES	0x40	250000	0x0	Yes
[0x41] LS_MAB_ALLOCATES_BY_TYPE	0x41	50000	0xb	Yes

Session Name	Creation Date
AMDuProf-EBP-ScimarkStable-Feb-17-2021_22-11-42	17th Feb 2021 | 10:11:42 ...
AMDuProf-EBP-ScimarkStable-Feb-17-2021_09-20-01	17th Feb 2021 | 9:20:01 AM

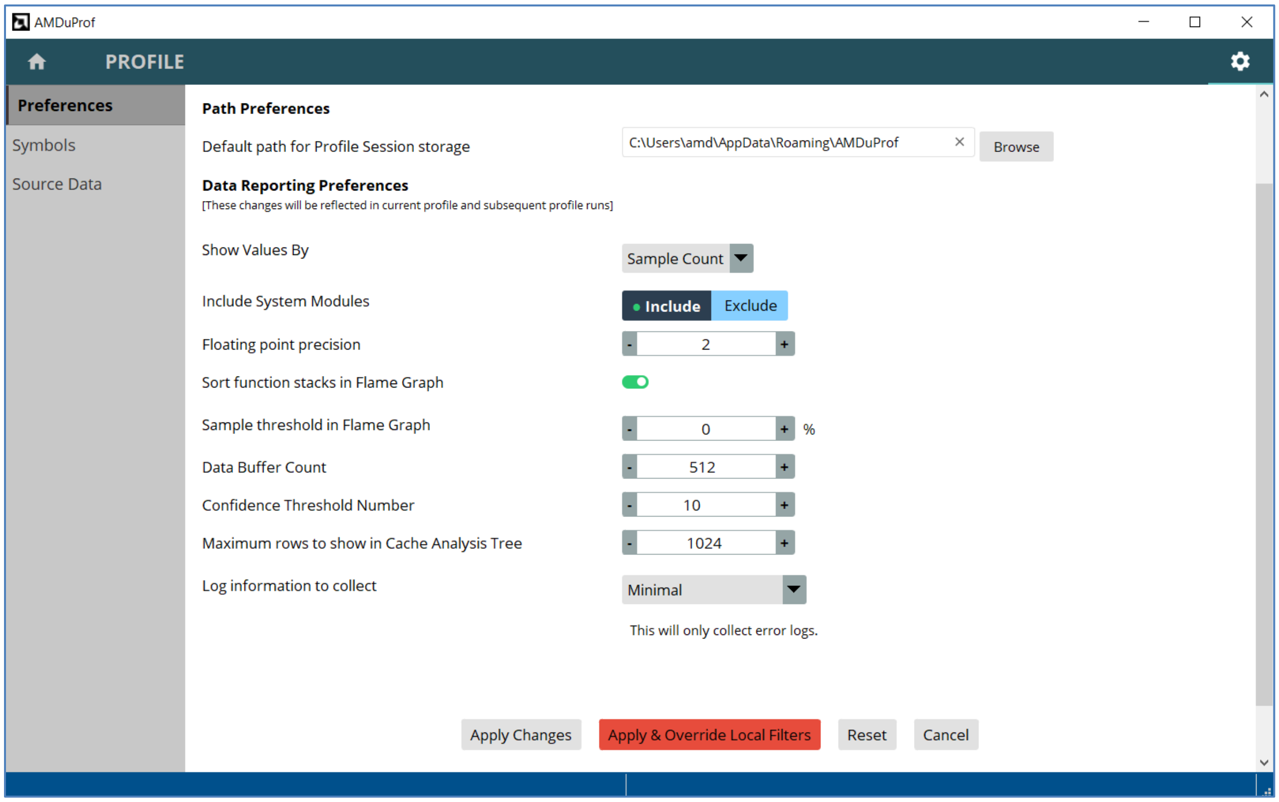
Saved Configurations

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 providing a config name in the bottom left corner (**Config Name** line-edit) in **PROFILE → Start Profiling**.

4.8 Settings

There are certain application-wide settings to customize the experience. The **SETTINGS** page is in top-right corner and is divided into three sections – **Preferences**, **Symbols** and **Source Data** each having a short description of what it contains.

Preferences: use this section to set the global path and data reporting preferences.


AMDuProf

PROFILE

Preferences
Symbols
Source Data

Path Preferences

Default path for Profile Session storage: C:\Users\amd\AppData\Roaming\AMDuProf [Browse]

Data Reporting Preferences
[These changes will be reflected in current profile and subsequent profile runs]

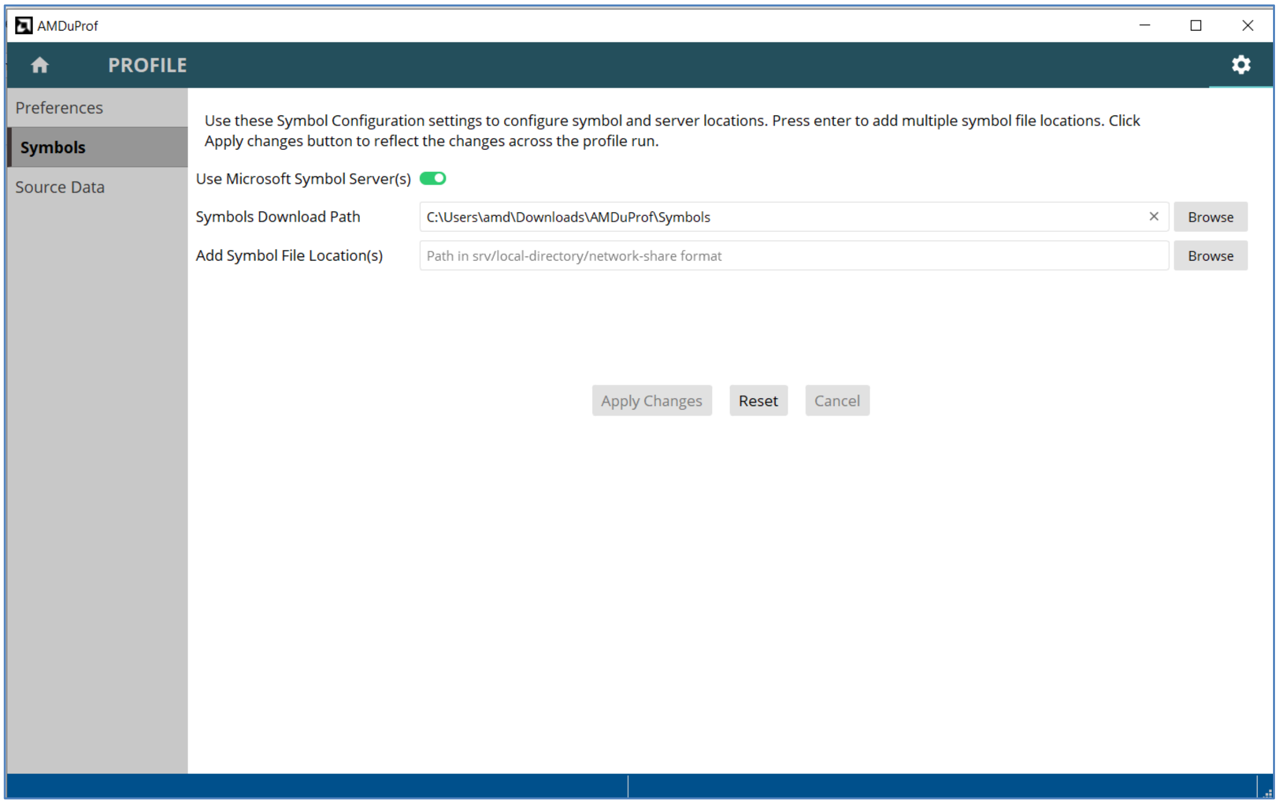
Show Values By: Sample Count
Include System Modules: [✓ Include] Exclude
Floating point precision: 2
Sort function stacks in Flame Graph: On
Sample threshold in Flame Graph: 0 %
Data Buffer Count: 512
Confidence Threshold Number: 10
Maximum rows to show in Cache Analysis Tree: 1024
Log information to collect: Minimal
This will only collect error logs.

SETTINGS – Preference

- The settings once changed can be applied by clicking the **Apply Changes** button. There are settings which are common with profile data filters and hence any change in them when applied through **Apply Changes** 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.

Symbols: use this section to configure the Symbol Paths and Symbol Server locations. The Symbol server is a Windows only option.

Source Data: use this section to set the Source view preferences.



AMDuProf

PROFILE

Preferences
Symbols
Source Data

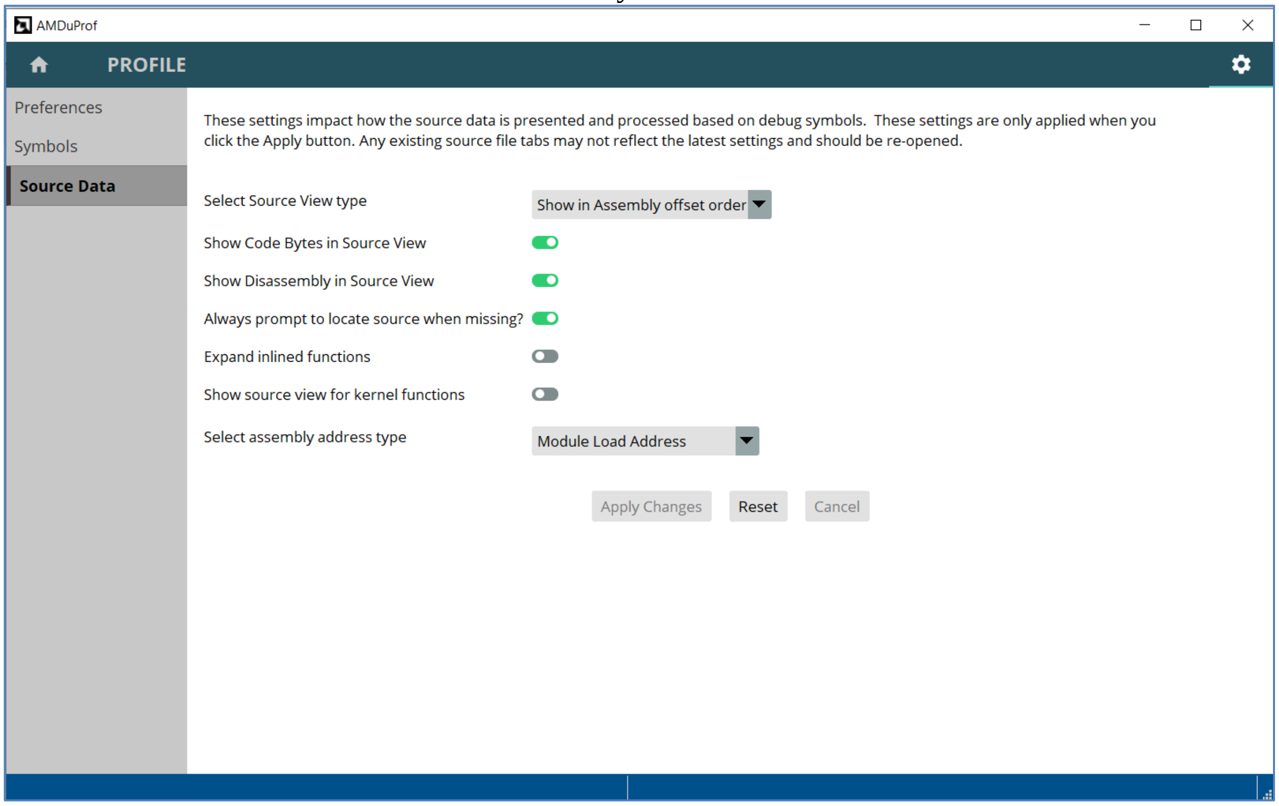
Use these Symbol Configuration settings to configure symbol and server locations. Press enter to add multiple symbol file locations. Click Apply changes button to reflect the changes across the profile run.

Use Microsoft Symbol Server(s)

Symbols Download Path: C:\Users\amd\Downloads\AMDuProf\Symbols

Add Symbol File Location(s): Path in srv/local-directory/network-share format

SETTINGS – Symbols section



AMDuProf

PROFILE

Preferences
Symbols
Source Data

These settings impact how the source data is presented and processed based on debug symbols. These settings are only applied when you click the Apply button. Any existing source file tabs may not reflect the latest settings and should be re-opened.

Select Source View type: Show in Assembly offset order

Show Code Bytes in Source View

Show Disassembly in Source View

Always prompt to locate source when missing?

Expand inlined functions

Show source view for kernel functions

Select assembly address type: Module Load Address

SETTINGS – Source data

Chapter 5 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
translate	Process the raw profile datafile and generates the profile db
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, or

`/tmp/AMDuProf_Linux_x64_X.Y.ZZZ/bin/AMDuProfCLI` (if installed using tarfile)

FreeBSD:

Run `/tmp/AMDuProf_FreeBSD_x64_X.Y.ZZZ/bin/AMDuProfCLI` binary.

5.1 How to start CPU profile?

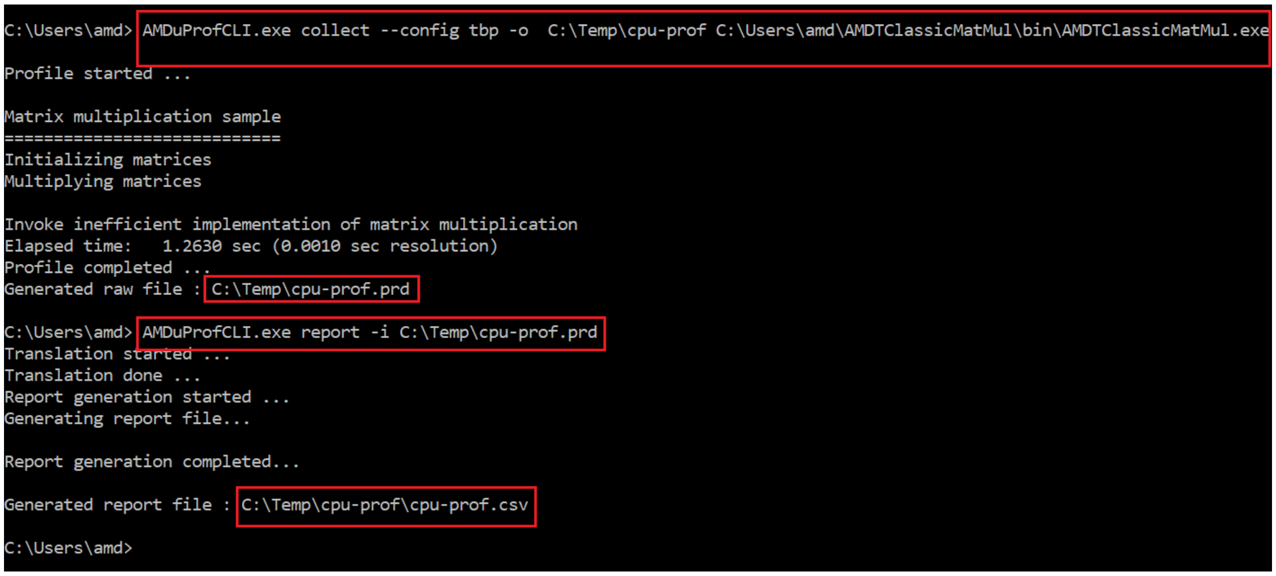
To profile and analyze the performance of a native (C/C++) application, you need to follow these steps:

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.

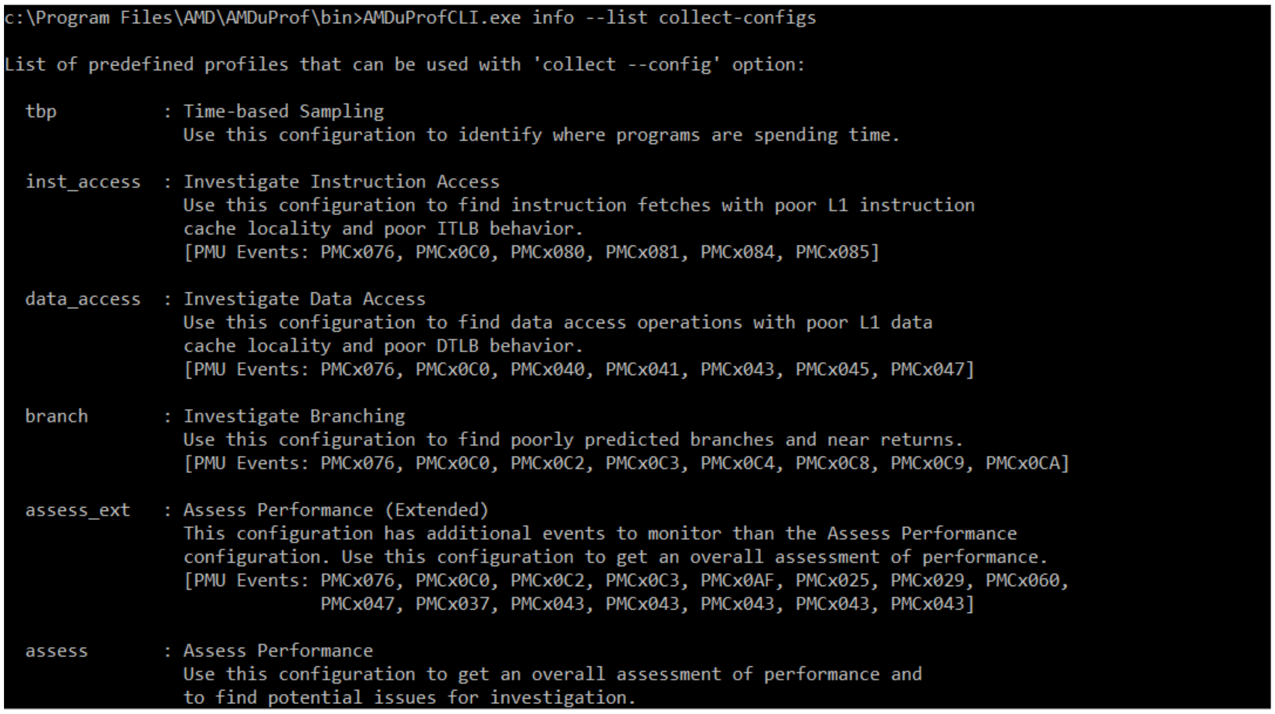
Note: On Linux, AMDuProfCLI **collect** command will generate a **caperf** file which will be passed as input file to **report** command.

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 info --list collect-configs
```

And the output will look like:



```
c:\Program Files\AMD\AMDuProf\bin>AMDuProfCLI.exe info --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]

  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, PMCx0C2, PMCx0C3, PMCx0AF, PMCx025, PMCx029, PMCx060,
                              PMCx047, PMCx037, PMCx043, PMCx043, PMCx043, PMCx043, PMCx043]

  assess       : Assess Performance
                 Use this configuration to get an overall assessment of performance and
                 to find potential issues for investigation.
```

AMDuProfCLI - list supported predefined configurations

Profile report

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

- EXECUTION – information about the target launch 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

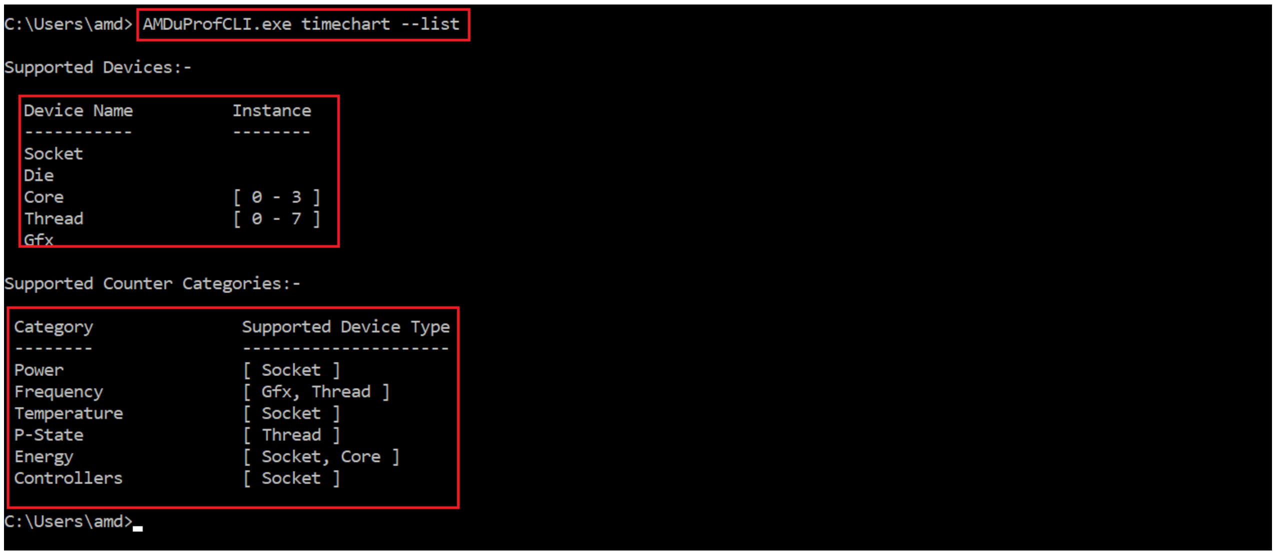
5.2 How to start Power profile?

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

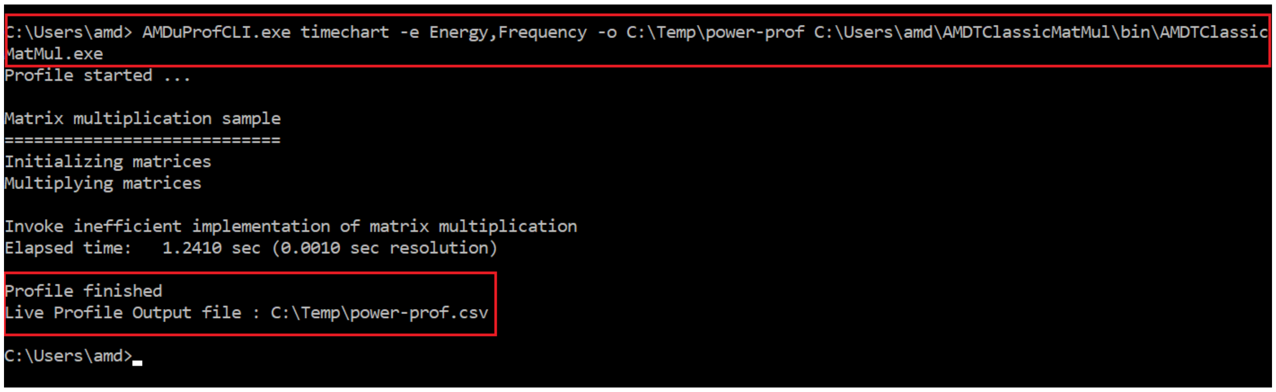
The timechart run to list the supported counter categories:



```
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.

5.3 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] [<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 <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.
`--config <config>`	Predefined sampling configuration to be used to collect samples. Use the command **info --list collect-configs** to get the list of supported configs.

`-e | --event <predefined-event>`
 	
`-e | --event <EVENT>`
	A <predefined event> is the symbolic name of a core PMC event that can be directly used with -e, --event.
	
	Use command '**info --list predefined-events**' for the list of supported predefined events.
	
	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 Core PMC event id.
	
	umask=<unit-mask>
	
	user=<0 | 1>

	os=<0 | 1> **interval**=<sampling interval> **ibsop-count-control**=<0 | 1> **call-graph** **loadstore** Ex: **-e event=pmcx76,interval=250000** Use command **info --list pmu-events** for the list of supported PMC 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. **call-graph** – To enable callstack data collection for this event. **loadstore** – To collect only the load and store IBS OP samples on Windows. Multiple occurrences of --event (-e) are allowed.
`-p | --pid <PID...>`	Profile existing processes (processes to attach to). Process IDs are separated by comma.
`--tid <TID...>`	Profile existing threads (threads to attach to). Threads IDs are separated by comma. This is a Linux only option.
`-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.*
`--interval <count>`	Sampling interval for PMC events. Note: This interval will override the sampling interval specified with individual events.
`-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

`--ip <IP Addr>`	IP address of the target system. Note: To perform remote profiling on a target system, remote agent AMDRemoteAgent should be launched first on the target system.
`--port <port>`	The port on which the remote agent AMDRemoteAgent is listening on the target system. Note: To perform remote profiling on a target system, remote agent AMDRemoteAgent should be launched first on the target system.

Windows specific options:

Option	Description
`--call-graph <I:D:S:F>`	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.
`-g`	Same as passing **--call-graph 1:128:user:nofpo**
`--data-buffer-count <count>`	To specify the number of data buffers to be used by the Windows uProf driver. This will help to reduce the missing samples.

Linux specific options:

Option	Description

`--call-graph <F:N>`	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`	Same as passing **--call-graph nofpo**
`--tid <TID,..>`	Profile existing threads(threads to attach to). Thread IDs are separated by comma.
`-m, --mmap-pages <size>`	Set kernel memory mapped data buffer to size. Size can be specified in pages or with a suffix Bytes(B/b), Kilo bytes(K/k), Megabytes(M/m), Gigabytes(G/g).
`--omp`	Profile OpenMP application. Note: 1. Applicable to per process and attach process profiling. 2. Not applicable to: a. System wide profiling b. Java app profiling 3. Compile the OpenMP application with LLVM/Clang 8.0 or later. Supported base languages: C, C++, Fortran
`--mpi`	Use this option to get the MPI profiling information.
`-O, --output-dir <directory name>`	Name of the output directory. This option should be used with –mpi option where the multiple raw data files are saved in a single directory.

Examples

Windows:

- Launch application AMDTClassicMatMul.exe and collect samples for CYCLES_NOT_IN_HALT and RETIRED_INST events:

```
C:\> AMDuProfCLI.exe collect -e cycles-not-in-halt -e retired-inst
--interval 1000000 -o c:\Temp\cpuprof-custom AMDTClassicMatMul.exe

C:\> AMDuProfCLI.exe collect -e event=cycles-not-in-halt,interval=250000
-e event=retired-inst,interval=500000 -o c:\Temp\cpuprof-custom
 AMDTClassicMatMul.exe
```

- 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
```

Linux:

- Launch application AMDTClassicMatMul.exe and collect samples for CYCLES_NOT_IN_HALT and RETIRED_INST events:

  ```
  $ ./AMDuProfCLI collect -e cycles-not-in-halt -e retired-inst
  --interval 1000000 -o /tmp/cpuprof-custom AMDTClassicMatMul-bin

  C:\> AMDuProfCLI.exe collect -e event=cycles-not-in-halt,interval=250000
  -e event=retired-inst,interval=500000 -o /tmp/cpuprof-custom
   AMDTClassicMatMul-bin
  ```

- 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
  ```

5.4 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] [<options>]
```

Common usages:

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

Options

Option	Description
`-h | --help`	Displays this help information on the console/terminal.
`-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.
`--detail`	Generate detailed report.
`--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 This option is applicable only with --detail option.
`-p, --pid <PID,..>`	Generate report for the specified PIDs. Process IDs are separated by comma.
`-g`	Print callgraph. Use with options --detail or --pid (-p). With --pid option, callgraph will be generated only if the callstack samples were collected for specified PIDs.

`--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 **info --list view-configs** to get the list of supported view-configs.
`--inline`	Show inline functions for C, C++ executables. Note: Using this option will increase the time taken to generate the report.
`--show-sys-src`	Generate detailed function report of the system module functions (if debug info is available) with source statements.
`--src-path <path1;...>`	Source file directories. (Semicolon separated paths.)
`--ascii event-dump`	To generate ASCII dump of IBS OP sample records from the given raw profile file.
`--disasm`	Generate detailed function report with assembly instructions.
`-s | --sort-by <EVENT>`	Specify the Timer, PMU, or IBS event on which the reported profile data will be sorted with arguments in the form of comma separated key=value pairs. Supported keys are: **event**=<timer | ibs-fetch | ibs-op | pmcxNNN> where NNN is hexadecimal Core PMC event id. **umask**=<unit-mask> **user**=<0 | 1> **os**=<0 | 1> Use command **info --list pmu-events** for the list of supported PMC events. Details about the arguments: **umask** - Unit mask in decimal or hexadecimal. Applicable only to PMU events. **user, os** - User, OS mode. Applicable only to PMU events.

	Multiple occurrences of –sort-by (-s) are **not** allowed.
`--imix`	Generate Instruction MIX report.
`--ignore-system-module`	Ignore samples from system modules.
`--show-percentage`	Show percentage of samples, instead of actual samples.
`--show-sample-count`	Show the number of samples. This option is enabled by default.
`--show-event-count`	Show number of events occurred
`--bin-path <path>`	Binary file path. Multiple use of --bin-path is allowed.
`--symbol-path <path1;...>`	Debug Symbol paths. (Semicolon separated paths.)
`-v | --verbose <n>`	Specify debug log messaging level. Valid values are: 1 : INFO 2 : DEBUG 3 : EXTENSIVE
`--ip <IP Addr>`	IP address of the target system. Note: To perform remote profiling on a target system, remote agent AMDRemoteAgent should be launched first on the target system.
`--port <port>`	The port on which the remote agent AMDRemoteAgent is listening on the target system. Note: To perform remote profiling on a target system, remote agent AMDRemoteAgent should be launched first on the target system.

Windows specific options

Option	Description
`--symbol-server <path1;...>`	Symbol Server directories. (Semicolon separated paths.)
`--symbol-cache-dir <path>`	Path to store the symbol files downloaded from the Symbol Servers.

Linux specific options

Option	Description
`-I, --input-dir <directory name>`	Input directory name. This is used to specify the data collected using ---mpi option and the directory specified should be the one specified with –output-dir option.
`--host <hostname>`	This option is used along with the --input-dir option. Generate report belonging to a specific host. Supported options are: <hostname>: Report process belonging to a specific host. all: Report all processes Note: If --host is not used then only the processes belonging to the system from which report is generated is reported.
`--limit-cacheinfo <n>`	Cut-off limit for entries in the cache line analysis report sections. Default value is 10.

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
  ```

5.5 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
`-h | --help`	Displays this help information.
`--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. **cac**: Collect all available cac counters. **controllers**: Collect all available controllers counters. Note: Multiple occurrences of -e is allowed.
`-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.

`--ip <IP Addr>`	IP address of the target system. Note: To perform remote profiling on a target system, remote agent AMDRemoteAgent should be launched first on the target system.
`--port <port>`	The port on which the remote agent AMDRemoteAgent is listening on the target system. Note: To perform remote profiling on a target system, remote agent AMDRemoteAgent should be launched first on the target system.

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
```

5.6 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
`-h | --help`	Displays the help information.
`--list <type>`	Lists the supported items for the following types: **collect-configs**: Predefined profile configurations that can be used with **collect** command's **--config** option. **predefined-events**: List of the supported predefined events that can be used with 'collect --event' option **view-configs**: List the supported data view configurations that can be used with **report** command's **--view** option. **pmu-events**: Raw PMC events that can be used with **collect** command's **--event** option. **cacheline-events**: List of event aliases to be used with 'report --sort-by' option for cache analysis.
`--collect-config <name>`	Displays details of the given profile configuration used with **collect --config <name>** option. Use **info --list collect-configs** command for details about the supported profile configurations.
`--view-config <name>`	Displays details of the given view configuration used in report generation option **report --view <name>**.

	Use **info --list view-configs** command for details about the supported data view configurations.
`--pmu-event <event>`	Displays details of the given pmu event. Use command **info --list pmu-events** for the list of supported PMC events.
`--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.
`--disasm <binary-path>`	Displays disassembly of the given binary file.

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
  ```
- To print system info:
  ```
  C:\> AMDuProfCLI.exe info --system
  ```
- To print list of predefined events:
  ```
  C:\> AMDuProfCLI.exe info --list predefined-events
  ```
- To print list of predefined profiles:
  ```
  C:\> AMDuProfCLI.exe info --list collect-configs
  ```
- To print list of PMU events:
  ```
  C:\> AMDuProfCLI.exe info --list pmu-events
  ```
- To print list of predefined report views:
  ```
  C:\> AMDuProfCLI.exe info --list view-configs
  ```
- To print details of predefined profile like “assess_ext”:

```
C:\> AMDuProfCLI.exe info --collect-config assess_ext
```

- To print details of the pmu-event like PMCx076:

  ```
  C:\> AMDuProfCLI.exe info --pmu-event pmcx76
  ```

- To print details of view configuration like ibs_op_overall:

  ```
  C:\> AMDuProfCLI.exe info --view-config ibs_op_overall
  ```

Chapter 6 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.

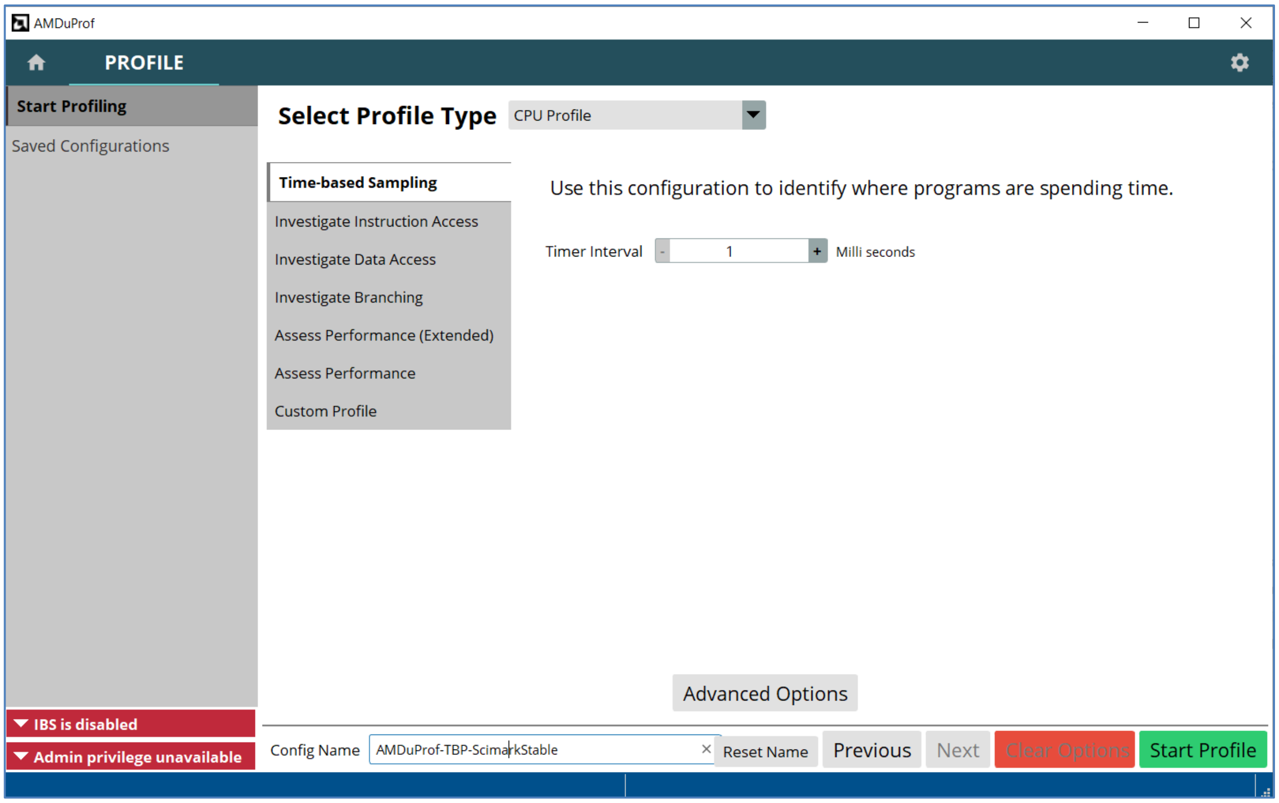
- **AMDuProfCLI**, the command-line-interface can be used to configure a profile run, collect the profile data, and generate the profile report.
  - **collect** option to configure and collect the profile data
  - **report** option to process the profile data and to generate the profile report
- **AMDuProf** 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
  - HPC - to analyze OpenMP profile data
  - Cache Analysis - to analyze the hot cache lines that are false shared
- Profile Control API to selectively enable and disable profiling from the target application by instrumenting it, to limit the scope of the profiling

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



Select Profile Type: CPU Profile

Use this configuration to identify where programs are spending time.

Timer Interval 1 Milli seconds

Config Name AMDuProf-TBP-ScimarkStable

Time based profile – configure

4. Click **Advanced Options** to enable Callstack, set symbol paths (if the debug files are in different locations) and other options. Refer *this* section for more information on this window.

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 see the **Hot Spots** window of **Summary** page. The hotspots are shown for **Timer** samples. Refer *this* section for more information on this window.

7. Clicking **ANALYZE** button on the top horizontal navigation bar will go to **Function HotSpots** window. Refer *this* section for more information on this window.

8. Clicking **ANALYZE** → **Metrics** will display the profile data table at various granularities - Process, Load Modules, Threads and Functions. Refer *this* section for more information on this window.

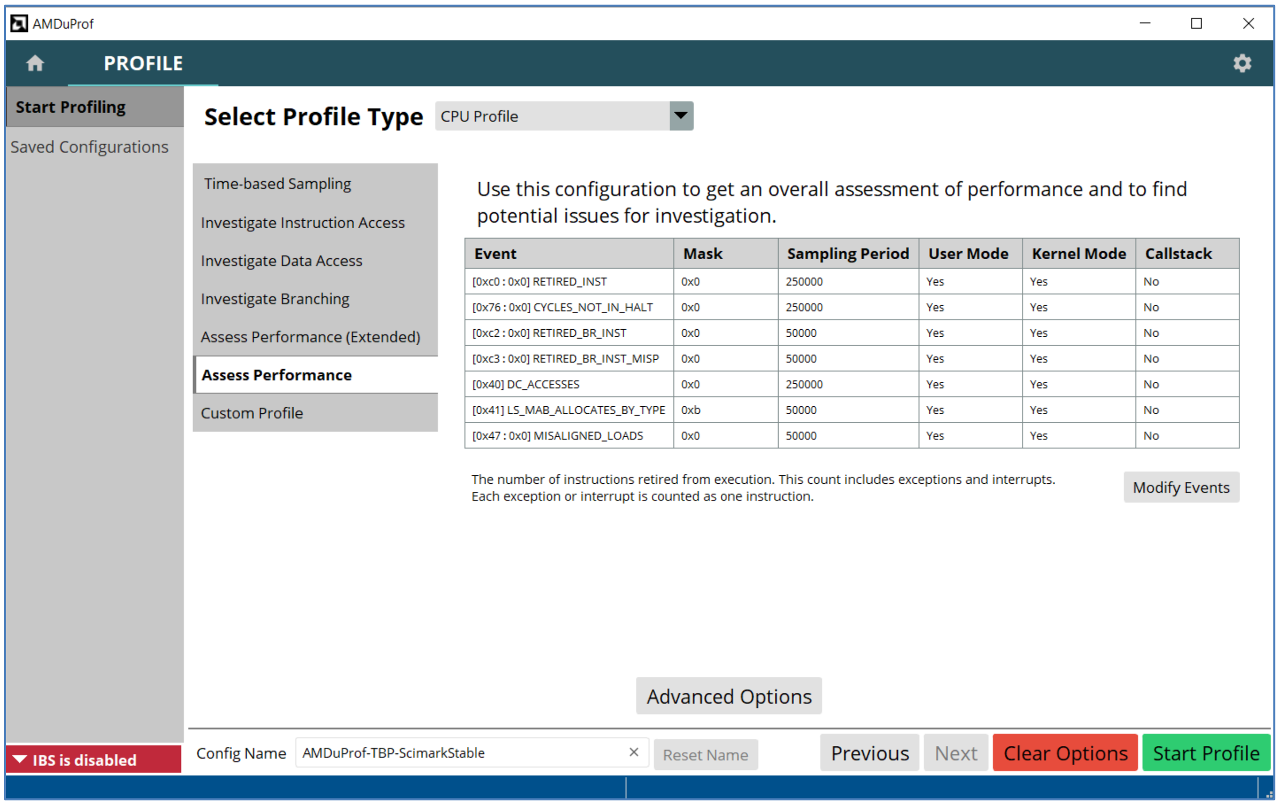
9. Double-clicking any entry on the **Functions** table in **Metrics** window will make the GUI load the source tab for that function in **SOURCES** page. Refer *this* section for more information on this window.

6.2 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. Clicking **PROFILE → Start Profiling** will navigate to the **Select Profile Target** window. After selecting the appropriate profile target, clicking **Next** button will take you to **Select Profile Type** fragment.
2. In **Select Profile Type** fragment, selecting **CPU Profile** from the drop-down list, will take you to the below screenshot.



Use this configuration to get an overall assessment of performance and to find potential issues for investigation.

Event	Mask	Sampling Period	User Mode	Kernel Mode	Callstack
[0xc0 : 0x0] RETIRED_INST	0x0	250000	Yes	Yes	No
[0x76 : 0x0] CYCLES_NOT_IN_HALT	0x0	250000	Yes	Yes	No
[0xc2 : 0x0] RETIRED_BR_INST	0x0	50000	Yes	Yes	No
[0xc3 : 0x0] RETIRED_BR_INST_MISP	0x0	50000	Yes	Yes	No
[0x40] DC_ACCESSES	0x0	250000	Yes	Yes	No
[0x41] LS_MAB_ALLOCATES_BY_TYPE	0xb	50000	Yes	Yes	No
[0x47 : 0x0] MISALIGNED_LOADS	0x0	50000	Yes	Yes	No

The number of instructions retired from execution. This count includes exceptions and interrupts. Each exception or interrupt is counted as one instruction.

Config Name AMDuProf-TBP-ScimarkStable

Event based profile - configure

3. Select **Assess Performance** in the left vertical pane as shown in the below screenshot. Refer *this* section for EBP based predefined sampling configurations.

4. Click **Advanced Options** to enable Callstack, set symbol paths (if the debug files are in different locations) and other options. Refer *this* section for more information on this window.

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.

7. Clicking **ANALYZE** button on the top horizontal navigation bar will go to **Function HotSpots** window. Refer *this* section for more information on this window.

8. Clicking **ANALYZE → Metrics** will display the profile data table at various granularities - Process, Load Modules, Threads and Functions. Refer *this* section for more information on this window.

9. 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.

6.3 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. Clicking **PROFILE → Start Profiling** will navigate to the **Select Profile Target** window. After selecting the appropriate profile target, clicking **Next** button will take you to **Select Profile Type** fragment.

2. In **Select Profile Type** fragment, select **CPU Profile** from the dropdown and then select **Instruction-Based Sampling** in the left vertical pane. Refer *this* section for predefined sampling configurations.

3. Click **Advanced Options** to enable Callstack, set symbol paths (if the debug files are in different locations) and other options. Refer *this* section for more information on this window.

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. Clicking **ANALYZE** button on the top horizontal navigation bar will go to **Function HotSpots** window. Refer *this* section for more information on this window.

7. Clicking **ANALYZE → Metrics** will display the profile data table at various granularities - Process, Load Modules, Threads and Functions. Refer *this* section for more information on this 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.

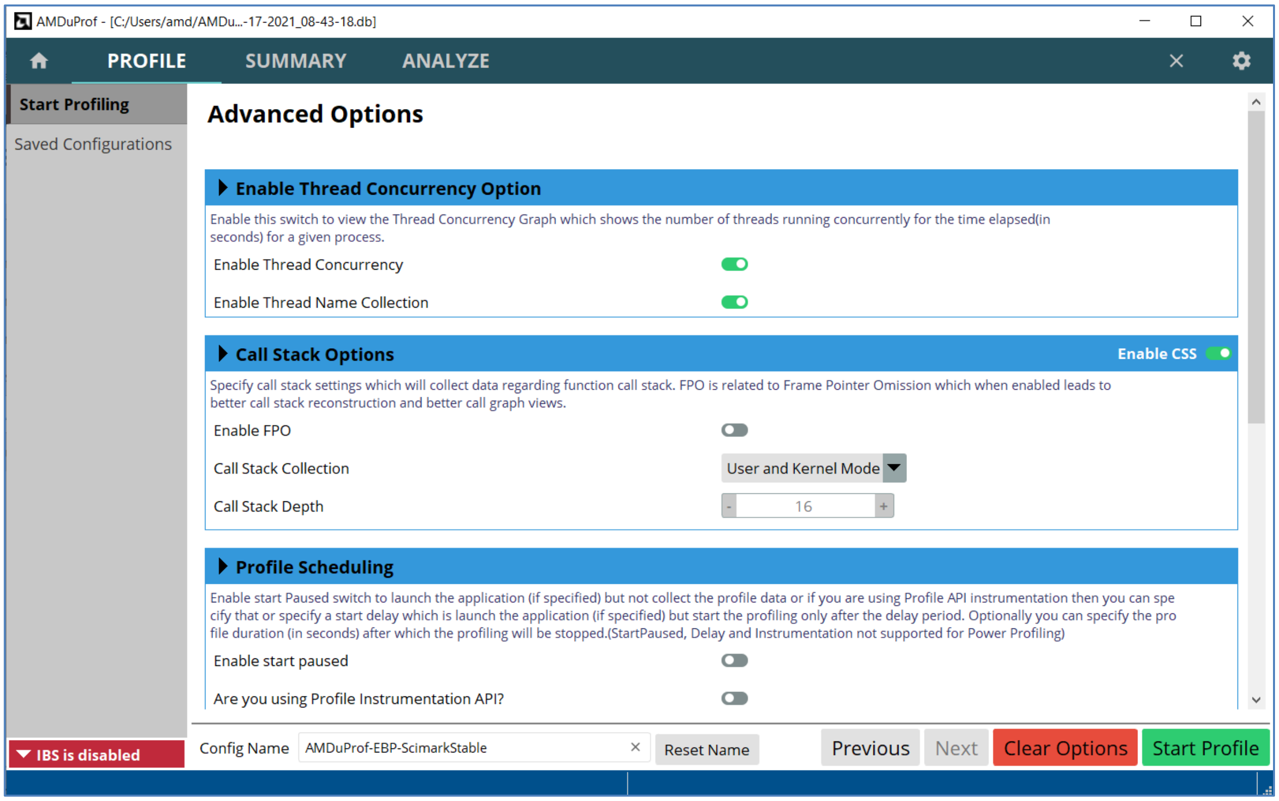
6.4 Analysis with Callstack samples

The callstack samples too can be collected for C, C++, and Java 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. Refer *this* section for more information on this window.

Note:

1. If the application is compiled with higher optimization levels and frame pointers are not emitted, then **Enable FPO** option can be enables. On Linux, this will increase the size of the raw profile file size.



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: on

Enable Thread Name Collection: on

Call Stack Options — Enable CSS: on

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: off

Call Stack Collection: User and Kernel Mode

Call Stack Depth: 16

Profile Scheduling

Enable start Paused switch to launch the application (if specified) but not collect the profile data or if you are using Profile API instrumentation then you can spe cify that or specify a start delay which is launch the application (if specified) but start the profiling only after the delay period. Optionally you can specify the pro file duration (in seconds) after which the profiling will be stopped.(StartPaused, Delay and Instrumentation not supported for Power Profiling)

Enable start paused: off

Are you using Profile Instrumentation API?: off

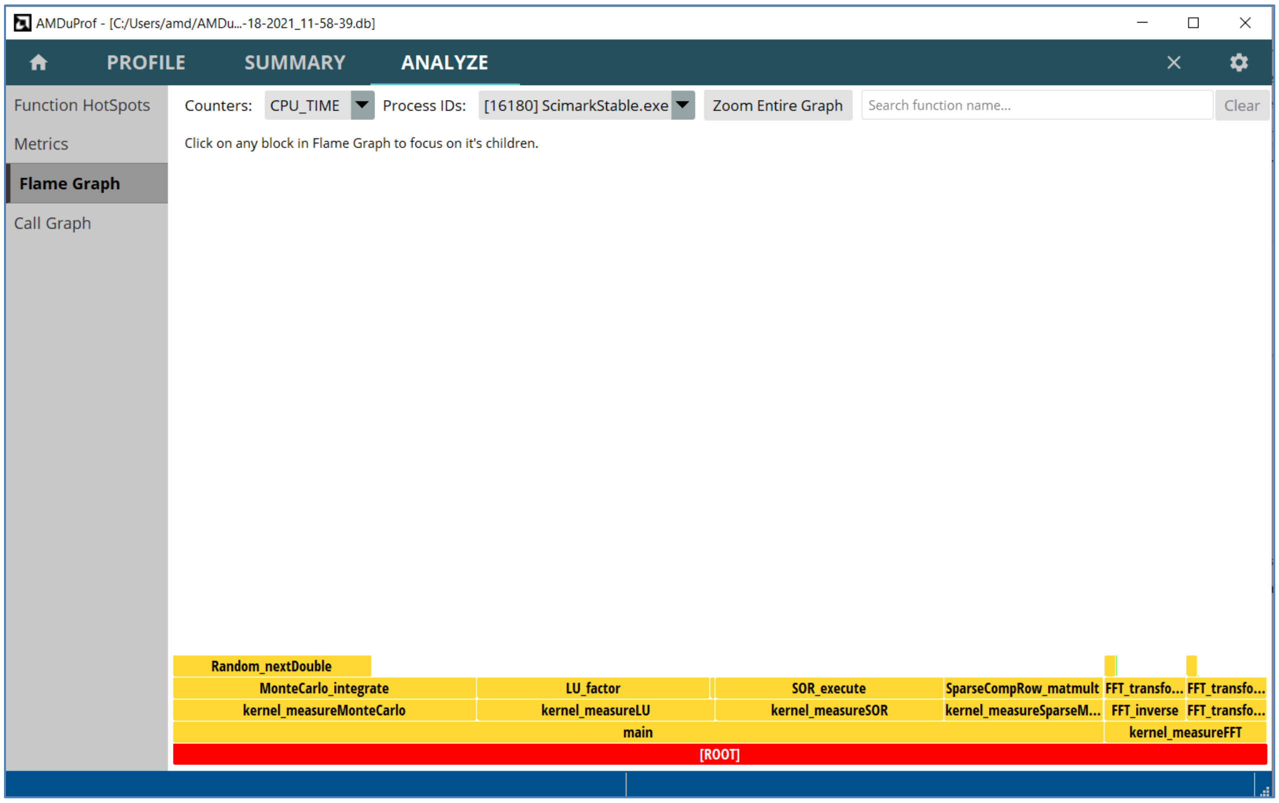
Config Name: AMDuProf-EBP-ScimarkStable

Start Profiling – Advanced Options

6.4.1 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.

Refer *this* section for more information on this window.



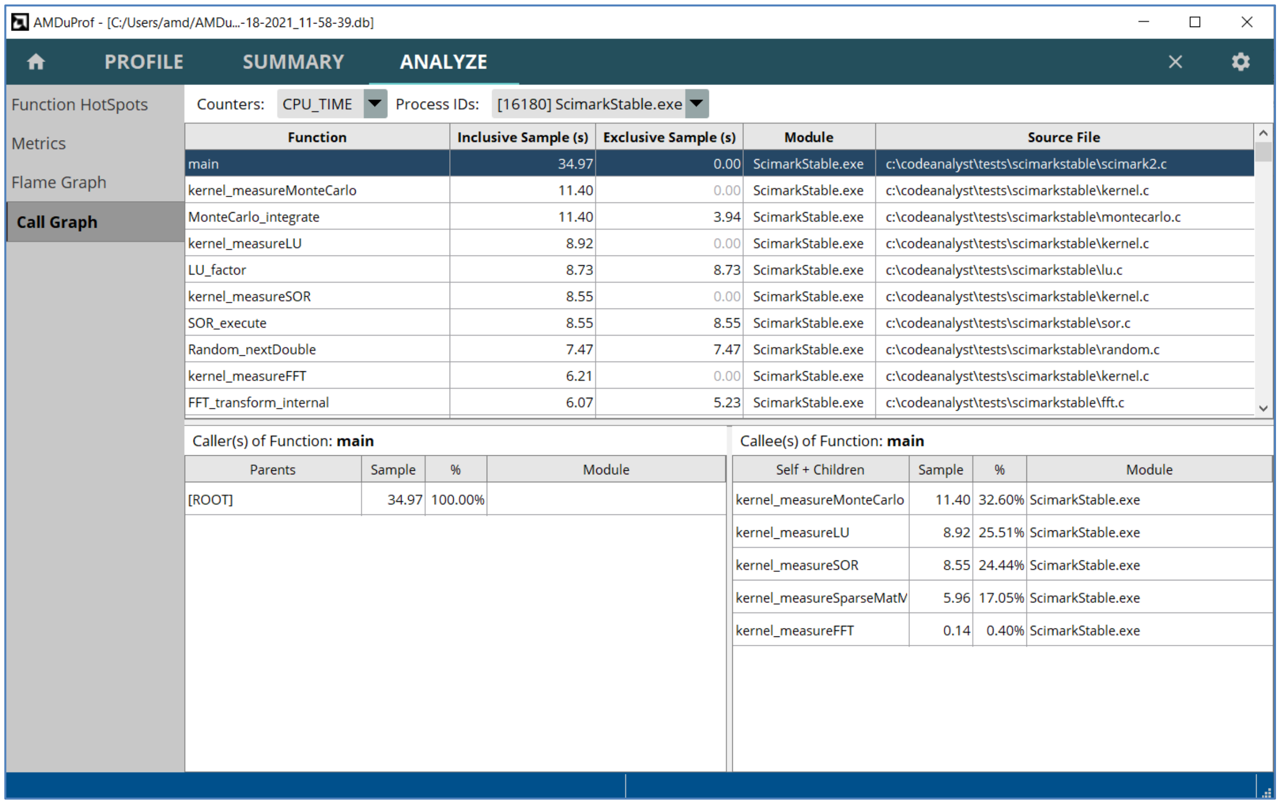
ANALYZE – Flame graph window

The Flamegraph can be displayed based on **Process IDs** and **Counters** dropdowns. It also has the function search box to search and highlight the given function name.

6.4.2 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.

Refer *this* section for more information on this window.



Counters: CPU_TIME Process IDs: [16180] ScimarkStable.exe

Function	Inclusive Sample (s)	Exclusive Sample (s)	Module	Source File
main	34.97	0.00	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\scimark2.c
kernel_measureMonteCarlo	11.40	0.00	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\kernel.c
MonteCarlo_integrate	11.40	3.94	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\montecarlo.c
kernel_measureLU	8.92	0.00	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\kernel.c
LU_factor	8.73	8.73	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\lu.c
kernel_measureSOR	8.55	0.00	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\kernel.c
SOR_execute	8.55	8.55	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\sor.c
Random_nextDouble	7.47	7.47	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\random.c
kernel_measureFFT	6.21	0.00	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\kernel.c
FFT_transform_internal	6.07	5.23	ScimarkStable.exe	c:\codeanalyst\tests\scimarkstable\fft.c

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

Parents	Sample	%	Module
[ROOT]	34.97	100.00%	

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

Self + Children	Sample	%	Module
kernel_measureMonteCarlo	11.40	32.60%	ScimarkStable.exe
kernel_measureLU	8.92	25.51%	ScimarkStable.exe
kernel_measureSOR	8.55	24.44%	ScimarkStable.exe
kernel_measureSparseMatM	5.96	17.05%	ScimarkStable.exe
kernel_measureFFT	0.14	0.40%	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.

6.5 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 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:

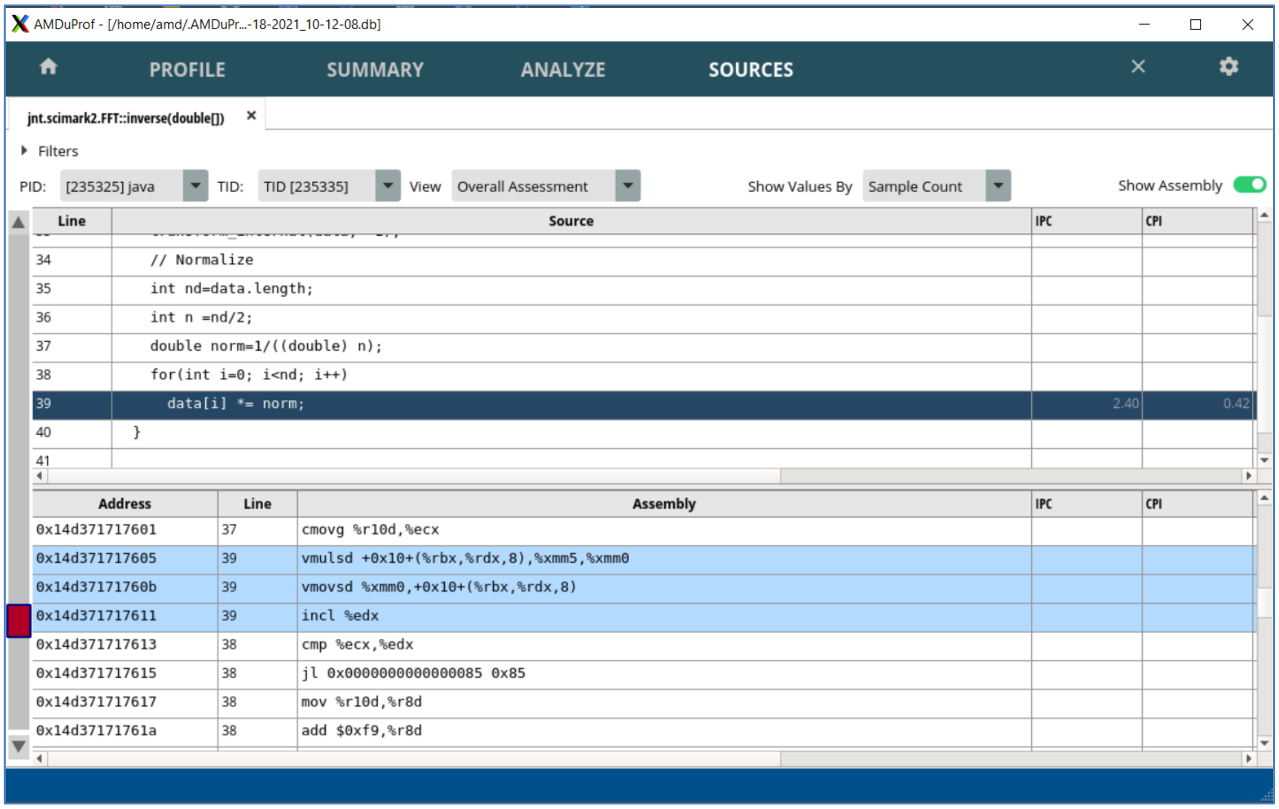
```
C:\> java -agentpath:
<C:\ProgramFiles\AMD\AMDuProf\bin\ProfileAgents\x64\AMDJvmtiAgent.dll>
<java-app-launch-options>
```

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.

Java source view

AMD uProf, will attribute the profile samples to Java methods and the source tab will show and the Java source lines with the corresponding samples attributed to them.

Refer *this* section for more information on source window.


AMDuProf - [/home/amd/.AMDuPr...-18-2021_10-12-08.db]
PROFILE
SUMMARY
ANALYZE
SOURCES
jnt.scimark2.FFT::inverse(double[])
Filters

PID: [235325] java TID: TID [235335] View Overall Assessment Show Values By Sample Count Show Assembly

Line	Source	IPC	CPI
34	// Normalize		
35	int nd=data.length;		
36	int n =nd/2;		
37	double norm=1/((double) n);		
38	for(int i=0; i<nd; i++)		
39	data[i] *= norm;	2.40	0.42
40	}		
41			

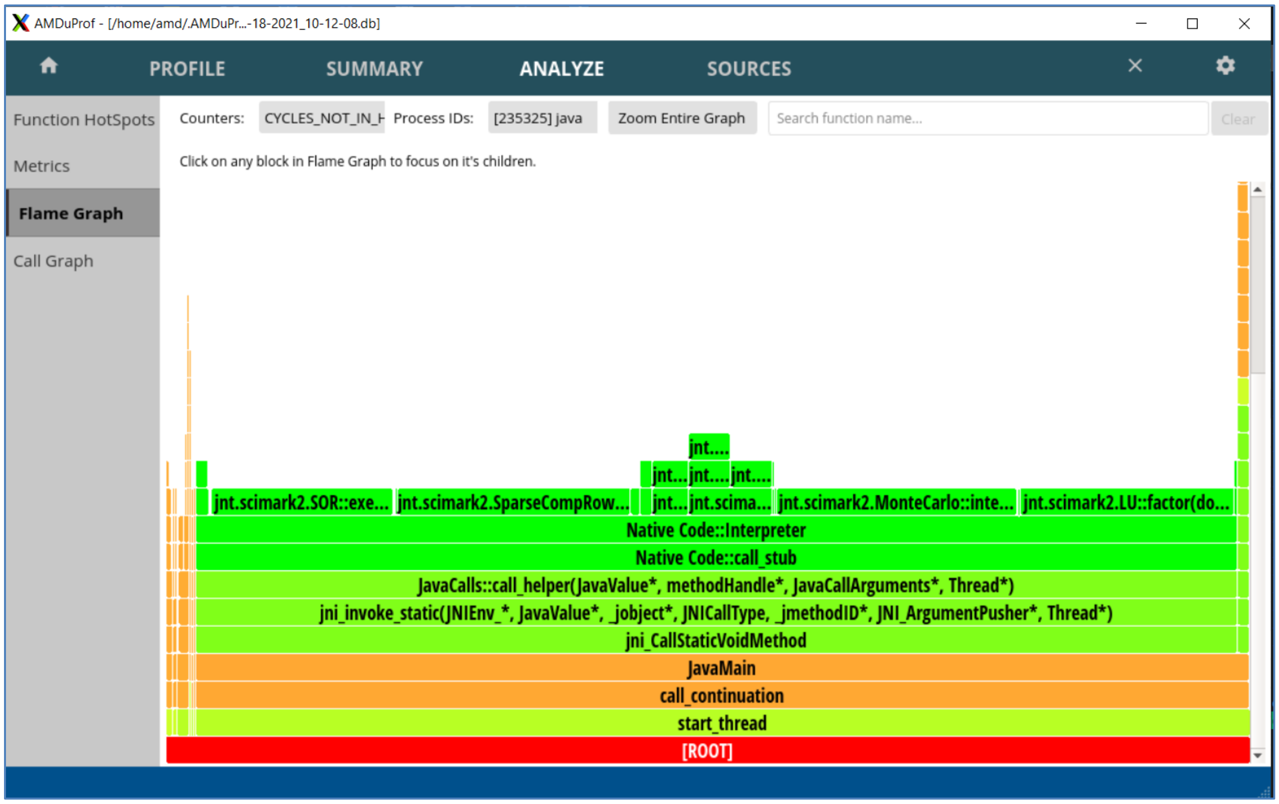
Address	Line	Assembly	IPC	CPI
0x14d371717601	37	cmovg %r10d,%ecx		
0x14d371717605	39	vmulsd +0x10+(%rbx,%rdx,8),%xmm5,%xmm0		
0x14d37171760b	39	vmovsd %xmm0,+0x10+(%rbx,%rdx,8)		
0x14d371717611	39	incl %edx		
0x14d371717613	38	cmp %ecx,%edx		
0x14d371717615	38	jl 0x0000000000000085 0x85		
0x14d371717617	38	mov %r10d,%r8d		
0x14d37171761a	38	add $0xf9,%r8d		

Java method – Source view

Java callstack profile

To collect callstack for profile java application, use the following command:

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


AMDuProf - [/home/amd/.AMDuPr...-18-2021_10-12-08.db]
PROFILE
SUMMARY
ANALYZE
SOURCES
Function HotSpots
Metrics
Flame Graph
Call Graph
Counters: CYCLES_NOT_IN_H
Process IDs: [235325] java
Zoom Entire Graph
Search function name...
Clear
Click on any block in Flame Graph to focus on it's children.
jnt....
jnt... jnt.... jnt....
jnt.scimark2.SOR::exe...
jnt.scimark2.SparseCompRow...
jnt... jnt.scima...
jnt.scimark2.MonteCarlo::inte...
jnt.scimark2.LU::factor(do...
Native Code::Interpreter
Native Code::call_stub
JavaCalls::call_helper(JavaValue*, methodHandle*, JavaCallArguments*, Thread*)
jni_invoke_static(JNIEnv_*, JavaValue*, _jobject*, JNICallType, _jmethodID*, JNI_ArgumentPusher*, Thread*)
jni_CallStaticVoidMethod
JavaMain
call_continuation
start_thread
[ROOT]

Java application – Flamegraph

Refer *this* section for more information on using Flamegraph window.

6.6 Cache Analysis

The **Cache Analysis** uses IBS OP samples to detect the hot false sharing cache lines in multithreaded and multi-process with shared memory applications.

At high level, this will feature will report

- The cache lines where there is a potential false sharing
- Offsets where those accesses occur and readers and writers to those offsets
- PID, TID, Function Name, Source file, Line number for those reader and writers
- Load latency for the loads to those cache lines

6.6.1.1 Supported Metrics

Following IBS OP derived metrics are used to generate false cache sharing report:

Metric	Description
LOAD_STORE_COUNT	Total Loads and stores sampled

LOAD_COUNT	Total Loads
STORE_COUNT	Total Stores
LOAD_LATENCY	Accumulated load latencies for the loads to cache lines
DC_L2_HIT	Load operations hit in data cache or L2 cache
LCL_CACHE_HIT (M)	Loads that was serviced from the local cache (L3) and the cache hit state was Modified.
LCL_CACHE_HIT (O)	Loads that was serviced from the local cache (L3) and the cache hit state was Owned.
LCL_CACHE_MISS	Loads that are missed in local cache (L3) and serviced by remote cache, local or remote DRAM.
RMT_CACHE_HIT (M)	Loads that was serviced from the remote cache (L3) and the cache hit state was Modified.
RMT_CACHE_HIT (O)	Loads that was serviced from the remote cache (L3) and the cache hit state was Owned.
DRAM_HIT_LCL	Loads that hit in local memory (Memory channels attached to local socket or local CCD)
DRAM_HIT_RMT	Loads that hit in remote memory (Memory channels attached to remote socket or other CCDs in the local socket)
STORE_DC_MISS	Store operations missed in data cache

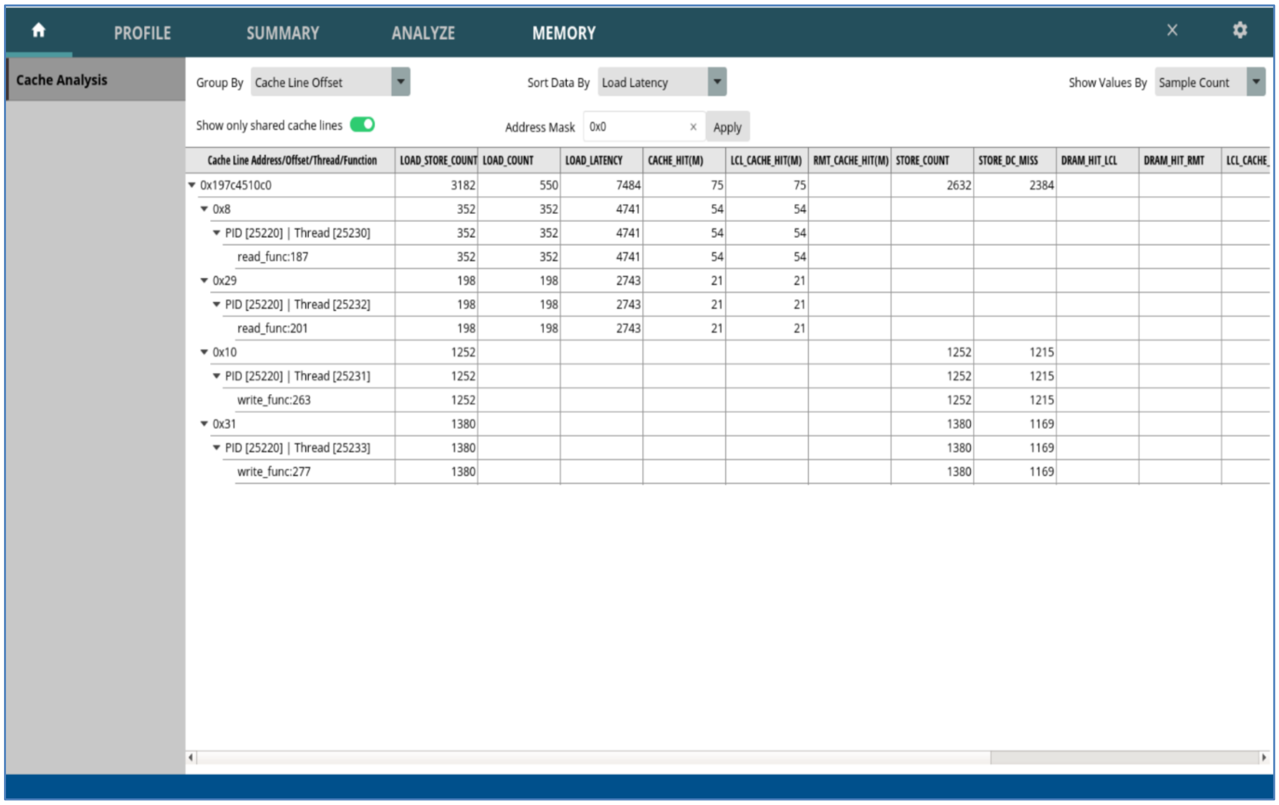
6.6.2 Cache Analysis using GUI

To configure and start profile:

To perform cache analysis, after selecting profile target select **Cache Analysis** profile type in **Select Profile Type** page and start the profile.

Analyzing the report:

After the profile completion, navigate to **Cache Analysis** page in **MEMORY** tab to analyze the profile data. This page shows th**e** cache-lines, and it offsets with the associated metric values.



Group By: Cache Line Offset
Sort Data By: Load Latency
Show Values By: Sample Count
Show only shared cache lines
Address Mask: 0x0
Apply

Cache Line Address/Offset/Thread/Function	LOAD_STORE_COUNT	LOAD_COUNT	LOAD_LATENCY	CACHE_HIT(M)	LCL_CACHE_HIT(M)	RMT_CACHE_HIT(M)	STORE_COUNT	STORE_DC_MISS	DRAM_HIT_LCL	DRAM_HIT_RMT	LCL_CACHE_
▾ 0x197c4510c0	3182	550	7484	75	75		2632	2384			
▾ 0x8	352	352	4741	54	54						
▾ PID [25220] | Thread [25230]	352	352	4741	54	54						
read_func:187	352	352	4741	54	54						
▾ 0x29	198	198	2743	21	21						
▾ PID [25220] | Thread [25232]	198	198	2743	21	21						
read_func:201	198	198	2743	21	21						
▾ 0x10	1252						1252	1215			
▾ PID [25220] | Thread [25231]	1252						1252	1215			
write_func:263	1252						1252	1215			
▾ 0x31	1380						1380	1169			
▾ PID [25220] | Thread [25233]	1380						1380	1169			
write_func:277	1380						1380	1169			

Cache Analysis

- Double clicking on the function will navigate to source view of that function.
- **Show only shared cache lines** switch can be turned off to show all the cache-lines for which samples were collected.
- **Address Mask** can be used to filters samples shown based on the mask provided.
- **Sort Data By** lists the metrics based on which the entries can be sorted. By default, it is based on **Load Latency** metric.
- **Group By** dropdown option decides how the cache-line samples are grouped in the detailed table. It has the following options:
    - Cache Line Offset
    - Threads and Processes
- **Show Values By** dropdown will let you either show the value as sample count or in percentage.

6.6.3 Cache Analysis using CLI

Data Collection:

The CLI has a config type called "memory" to cache analysis data. Run the following command to collect the profile data:

```
$ AMDuProfCLI collect --config memory -o /tmp/cache_analysis <target app>
```

This command will launch the program and collect the profile data required to generate the cache analysis report. The data file `/tmp/cache_analysis.caperf` will contain raw profile data.

Report generation and Analysis:

Use the following CLI command to generate the cache analysis report

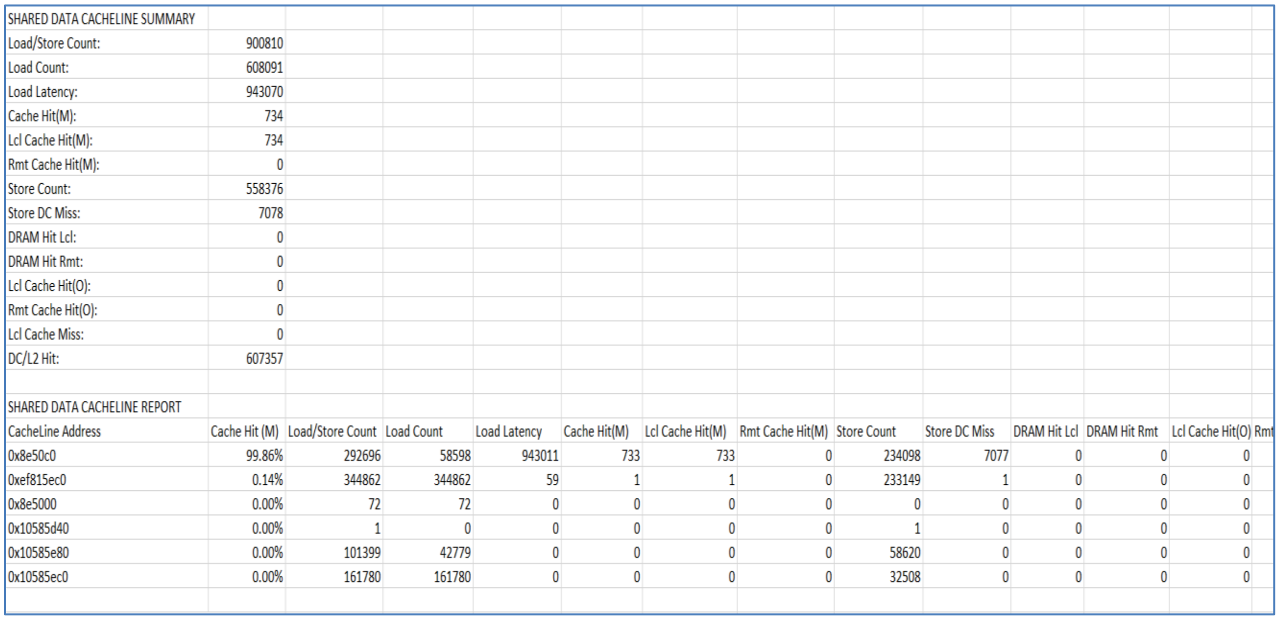
```
$ AMDuProfCLI report -i /tmp/cache_analysis.caperf
```

This will generate a CSV report at `/tmp/cache_analysis/cache_analysis.csv` and this report will have the following sections:

- **SHARED DATA CACHELINE SUMMARY**: Lists the summary values of all the metrics.
- **SHARED DATA CACHELINE REPORT**: Lists the cache lines and the associated summary values of the metrics.
- **SHARED DATA CACHELINE DETAIL REPORT:** Lists
    - The cache lines where there is a potential false sharing
    - Offsets where those accesses occur and readers & writers to those offsets
    - PID, TID, Function Name, Source file, Line number for those reader and writers
    - Load latency for the loads to those cache lines
    - Supported metrics

By default, the generated report will have a cutoff limit of 10 entries for each of the above-mentioned sections. To include more entries, use option `--limit-cacheinfo <cutoff-value>` with report command:

```
$ AMDuProfCLI report  --limit-cacheinfo <cutoff-value> -i /tmp/cache_analysis.caperf
```

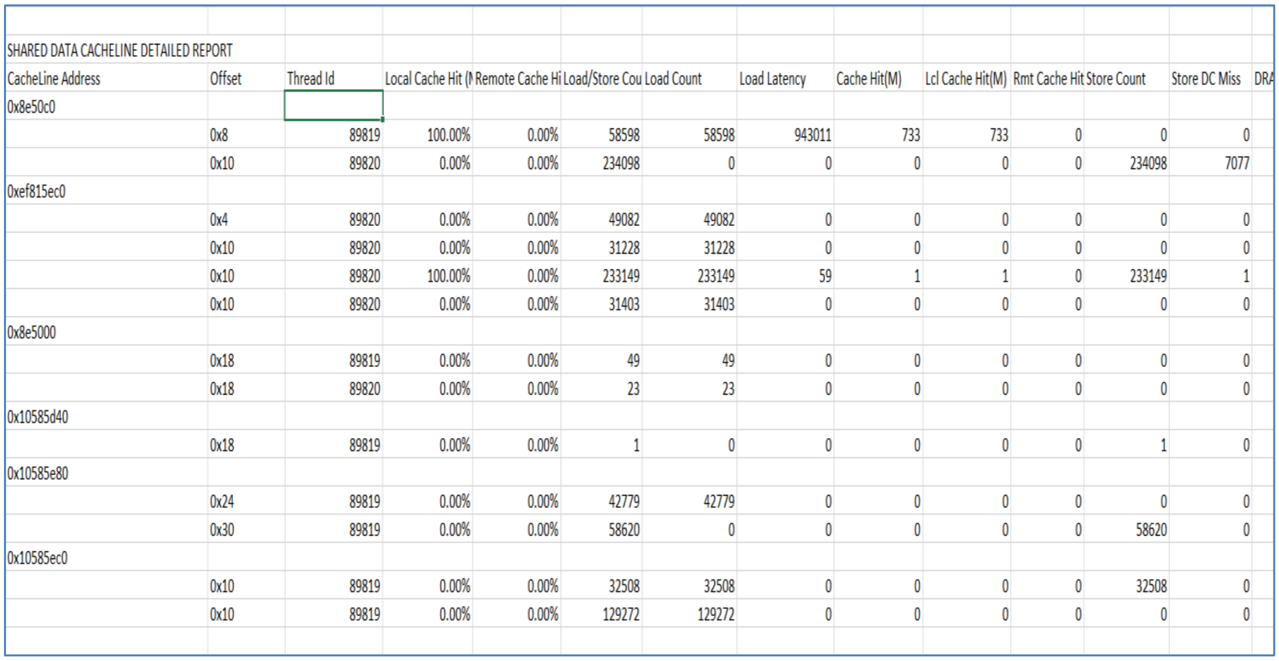


SHARED DATA CACHELINE SUMMARY	
Load/Store Count:	900810
Load Count:	608091
Load Latency:	943070
Cache Hit(M):	734
Lcl Cache Hit(M):	734
Rmt Cache Hit(M):	0
Store Count:	558376
Store DC Miss:	7078
DRAM Hit Lcl:	0
DRAM Hit Rmt:	0
Lcl Cache Hit(O):	0
Rmt Cache Hit(O):	0
Lcl Cache Miss:	0
DC/L2 Hit:	607357

SHARED DATA CACHELINE REPORT

CacheLine Address	Cache Hit (M)	Load/Store Count	Load Count	Load Latency	Cache Hit(M)	Lcl Cache Hit(M)	Rmt Cache Hit(M)	Store Count	Store DC Miss	DRAM Hit Lcl	DRAM Hit Rmt	Lcl Cache Hit(O)	Rmt
0x8e50c0	99.86%	292696	58598	943011	733	733	0	234098	7077	0	0	0	
0xef815ec0	0.14%	344862	344862	59	1	1	0	233149	1	0	0	0	
0x8e5000	0.00%	72	72	0	0	0	0	0	0	0	0	0	
0x10585d40	0.00%	1	0	0	0	0	0	1	0	0	0	0	
0x10585e80	0.00%	101399	42779	0	0	0	0	58620	0	0	0	0	
0x10585ec0	0.00%	161780	161780	0	0	0	0	32508	0	0	0	0	

Cache Analysis - Summary sections



SHARED DATA CACHELINE DETAILED REPORT

CacheLine Address	Offset	Thread Id	Local Cache Hit (M	Remote Cache Hi	Load/Store Cou	Load Count	Load Latency	Cache Hit(M)	Lcl Cache Hit(M)	Rmt Cache Hit	Store Count	Store DC Miss	DRA
0x8e50c0													
	0x8	89819	100.00%	0.00%	58598	58598	943011	733	733	0	0	0	
	0x10	89820	0.00%	0.00%	234098	0	0	0	0	0	234098	7077	
0xef815ec0													
	0x4	89820	0.00%	0.00%	49082	49082	0	0	0	0	0	0	
	0x10	89820	0.00%	0.00%	31228	31228	0	0	0	0	0	0	
	0x10	89820	100.00%	0.00%	233149	233149	59	1	1	0	233149	1	
	0x10	89820	0.00%	0.00%	31403	31403	0	0	0	0	0	0	
0x8e5000													
	0x18	89819	0.00%	0.00%	49	49	0	0	0	0	0	0	
	0x18	89820	0.00%	0.00%	23	23	0	0	0	0	0	0	
0x10585d40													
	0x18	89819	0.00%	0.00%	1	0	0	0	0	0	1	0	
0x10585e80													
	0x24	89819	0.00%	0.00%	42779	42779	0	0	0	0	0	0	
	0x30	89819	0.00%	0.00%	58620	0	0	0	0	0	58620	0	
0x10585ec0													
	0x10	89819	0.00%	0.00%	32508	32508	0	0	0	0	32508	0	
	0x10	89819	0.00%	0.00%	129272	129272	0	0	0	0	0	0	

Cache Analysis – detailed report

By default, the metrics are sorted by **Cache Hit (M)** metric. Use any of the following metric with the `--sort-by` option to changes the sorting by order:

Sort-by metric	Description
ldst-count	Total Loads and stores sampled

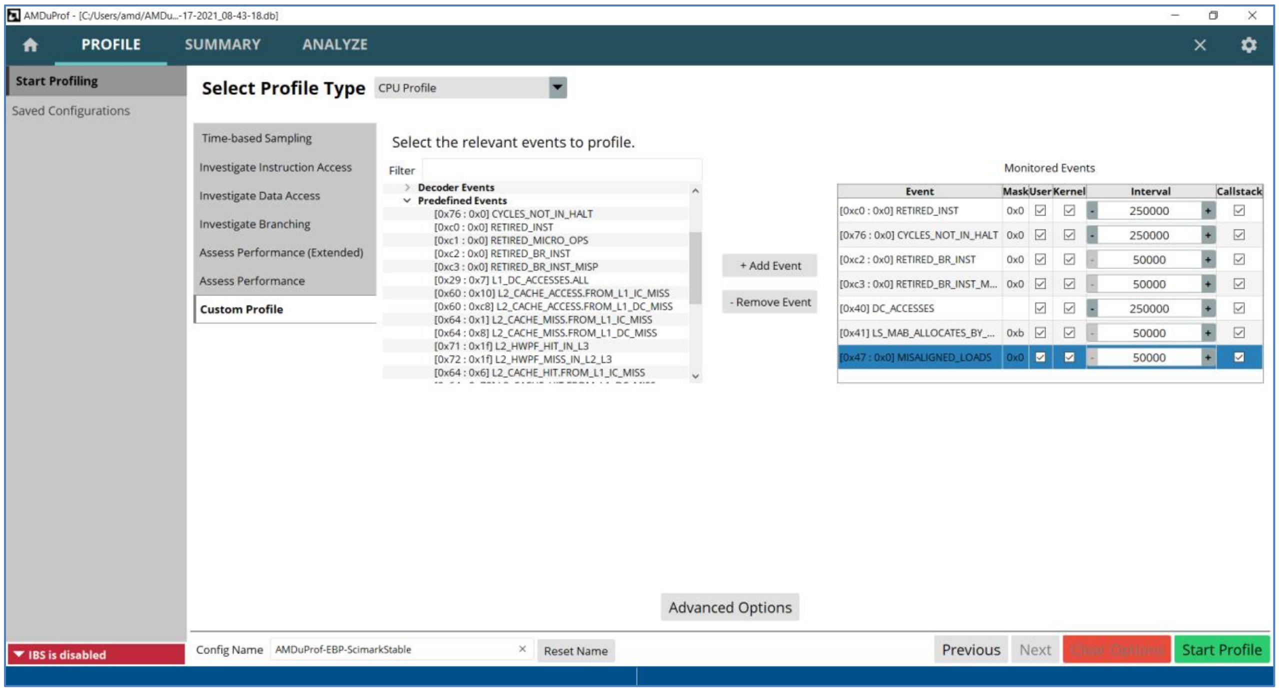
ld-count	Total Loads
st-count	Total Stores
cache-hitm	Loads that was serviced either from the local or remote cache (L3) and the cache hit state was Modified.
lcl-cache-hitm	Loads that was serviced from the local cache (L3) and the cache hit state was Modified.
rmt-cache-hitm	Loads that was serviced from the remote cache (L3) and the cache hit state was Modified.
lcl-dram-hit	Loads that hit in local memory (Memory channels attached to local socket or local CCD)
rmt-dram-hit	Loads that hit in remote memory (Memory channels attached to remote socket or other CCDs in the local socket)
l3-miss	Loads that are missed in local cache (L3) and serviced by remote cache, local or remote DRAM.
st-dc-miss	Store operations missed in data cache

6.7 Custom Profile

Apart the predefine configurations, the user can choose the interesting events to profile. To perform the custom profile, follow the steps mentioned here:

To configure and start profile:

1. Clicking **PROFILE → Start Profiling** will navigate to the **Select Profile Target** window. After selecting the appropriate profile target, clicking **Next** button will take you to **Select Profile Type** fragment.
2. In **Select Profile Type** fragment, selecting **CPU Profile** from the drop-down list, will take you to the below screenshot.
3. Select **Custom Profile** in the left vertical pane as shown in the below screenshot.



Select Profile Type: CPU Profile

Select the relevant events to profile.

Filter

- Decoder Events
- Predefined Events
  - [0x76 : 0x0] CYCLES_NOT_IN_HALT
  - [0xc0 : 0x0] RETIRED_INST
  - [0xc1 : 0x0] RETIRED_MICRO_OPS
  - [0xc2 : 0x0] RETIRED_BR_INST
  - [0xc3 : 0x0] RETIRED_BR_INST_MISP
  - [0x29 : 0x7] L1_DC_ACCESSES.ALL
  - [0x60 : 0x10] L2_CACHE_ACCESS.FROM_L1_IC_MISS
  - [0x60 : 0xc8] L2_CACHE_ACCESS.FROM_L1_DC_MISS
  - [0x64 : 0x1] L2_CACHE_MISS.FROM_L1_IC_MISS
  - [0x64 : 0x8] L2_CACHE_MISS.FROM_L1_DC_MISS
  - [0x71 : 0x1f] L2_HWPF_HIT_IN_L3
  - [0x72 : 0x1f] L2_HWPF_MISS_IN_L2_L3
  - [0x64 : 0x6] L2_CACHE_HIT.FROM_L1_IC_MISS

Monitored Events

Event	Mask	User	Kernel	Interval	Callstack
[0xc0 : 0x0] RETIRED_INST	0x0	✓	✓	250000	✓
[0x76 : 0x0] CYCLES_NOT_IN_HALT	0x0	✓	✓	250000	✓
[0xc2 : 0x0] RETIRED_BR_INST	0x0	✓	✓	50000	✓
[0xc3 : 0x0] RETIRED_BR_INST_M...	0x0	✓	✓	50000	✓
[0x40] DC_ACCESSES		✓	✓	250000	✓
[0x41] LS_MAB_ALLOCATES_BY_...	0xb	✓	✓	50000	✓
[0x47 : 0x0] MISALIGNED_LOADS	0x0	✓	✓	50000	✓

Config Name: AMDuProf-EBP-ScimarkStable

Custom Profile

4. Click **Advanced Options** to enable Callstack, set symbol paths (if the debug files are in different locations) and other options. Refer *this* section for more information on this window.

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.

7. Clicking **ANALYZE** button on the top horizontal navigation bar will go to **Function HotSpots** window. Refer *this* section for more information on this window.

8. Clicking **ANALYZE → Metrics** will display the profile data table at various granularities - Process, Load Modules, Threads and Functions. Refer *this* section for more information on this window.

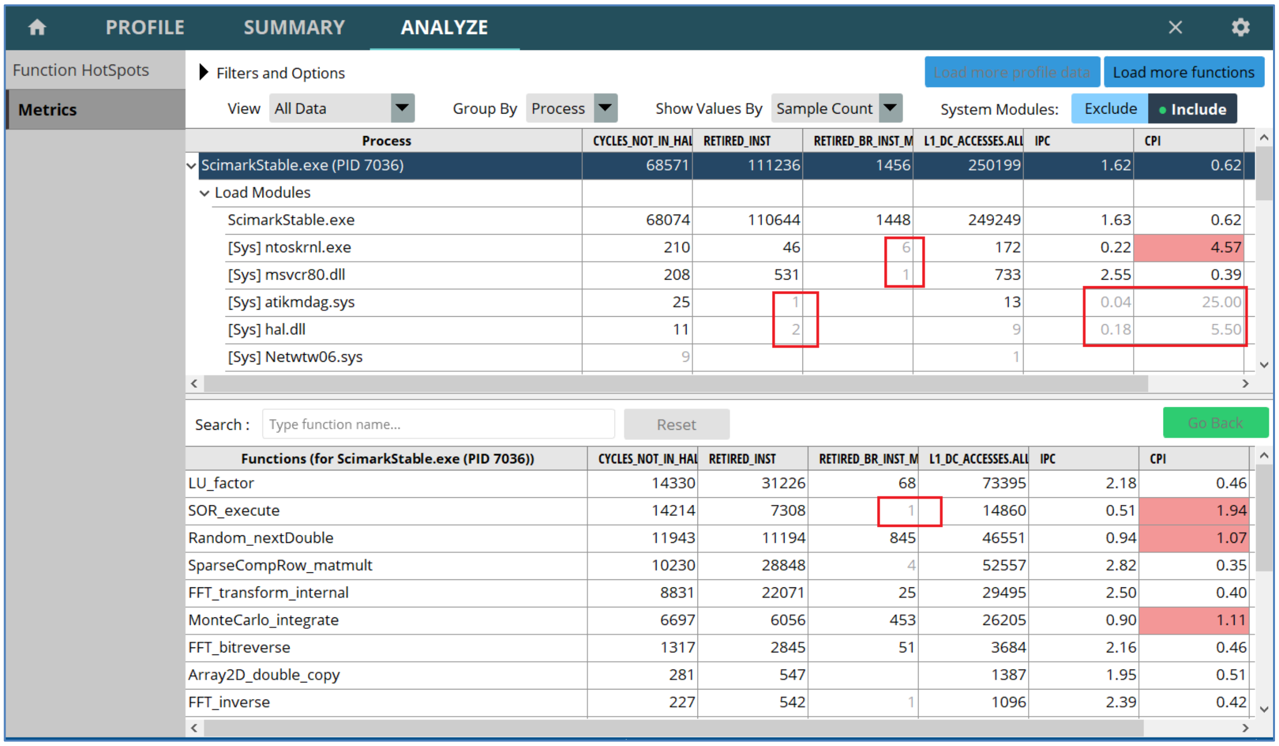
9. Double-clicking any entry on the **Functions** table in **Metrics** window will make the GUI load the source tab for that function in **SOURCES** page. Refer *this* section for more information on this window.

6.8 Advisory

Confidence Threshold

The metric with low number of samples collected for a program unit either due to multiplexing or statical sampling will be greyed out.

- This is applicable to SW Timer and Core PMC based metrics.
- This confidence threshold value can be set through **Preferences** section in **SETTINGS** page.



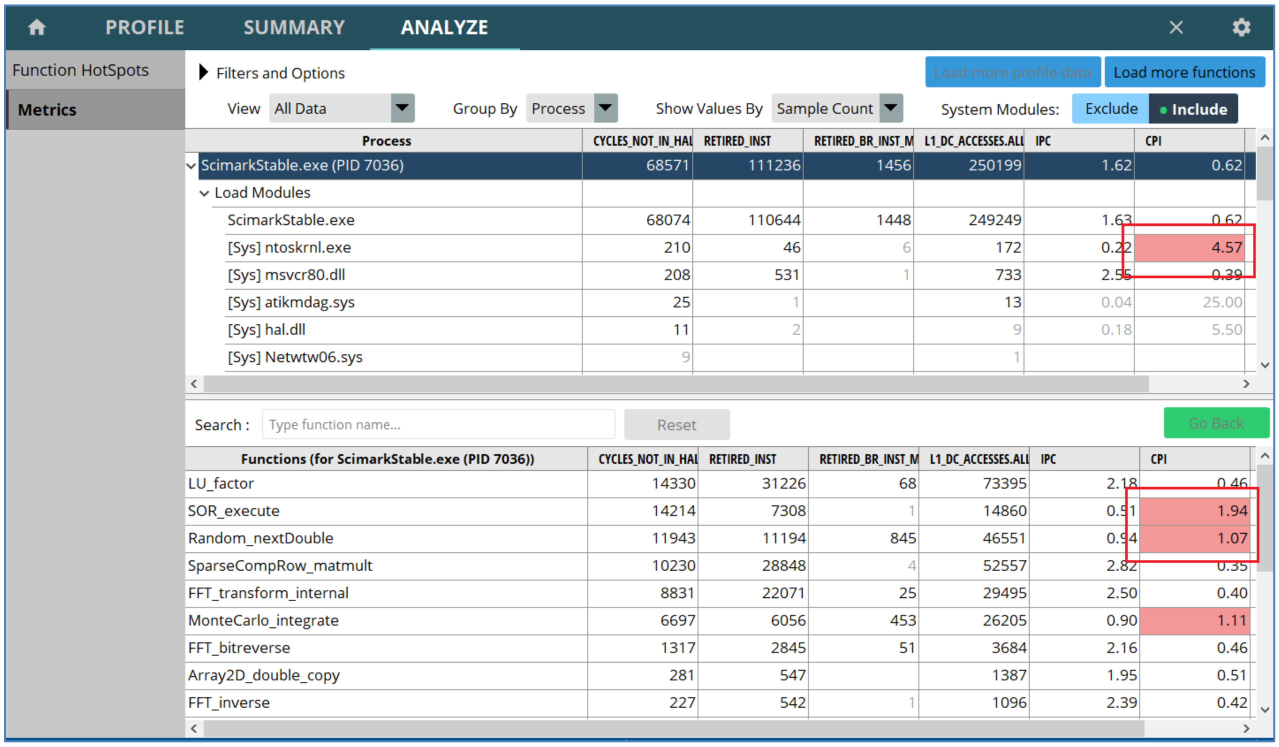
Process	CYCLES_NOT_IN_HAL	RETIRED_INST	RETIRED_BR_INST_M	L1_DC_ACCESSES.ALL	IPC	CPI
ScimarkStable.exe (PID 7036)	68571	111236	1456	250199	1.62	0.62
Load Modules						
ScimarkStable.exe	68074	110644	1448	249249	1.63	0.62
[Sys] ntoskrnl.exe	210	46	6	172	0.22	4.57
[Sys] msvcr80.dll	208	531	1	733	2.55	0.39
[Sys] atikmdag.sys	25	1		13	0.04	25.00
[Sys] hal.dll	11	2		9	0.18	5.50
[Sys] Netwtw06.sys	9			1		

Functions (for ScimarkStable.exe (PID 7036))	CYCLES_NOT_IN_HAL	RETIRED_INST	RETIRED_BR_INST_M	L1_DC_ACCESSES.ALL	IPC	CPI
LU_factor	14330	31226	68	73395	2.18	0.46
SOR_execute	14214	7308	1	14860	0.51	1.94
Random_nextDouble	11943	11194	845	46551	0.94	1.07
SparseCompRow_matmult	10230	28848	4	52557	2.82	0.35
FFT_transform_internal	8831	22071	25	29495	2.50	0.40
MonteCarlo_integrate	6697	6056	453	26205	0.90	1.11
FFT_bitreverse	1317	2845	51	3684	2.16	0.46
Array2D_double_copy	281	547		1387	1.95	0.51
FFT_inverse	227	542	1	1096	2.39	0.42

Confidence level of metrics – low confidence samples are greyed out

Issue Threshold

Highlight the CPI metric’s cells exceeding the specific threshold value (>1.0). Those cells will be highlighted in pink to show them as potential performance problem.



Process	CYCLES_NOT_IN_HAL	RETIRED_INST	RETIRED_BR_INST_M	L1_DC_ACCESSES.ALL	IPC	CPI
ScimarkStable.exe (PID 7036)	68571	111236	1456	250199	1.62	0.62
Load Modules						
ScimarkStable.exe	68074	110644	1448	249249	1.63	0.62
[Sys] ntoskrnl.exe	210	46	6	172	0.22	4.57
[Sys] msvcr80.dll	208	531	1	733	2.55	0.39
[Sys] atikmdag.sys	25	1		13	0.04	25.00
[Sys] hal.dll	11	2		9	0.18	5.50
[Sys] Netwtw06.sys	9			1		

Functions (for ScimarkStable.exe (PID 7036))	CYCLES_NOT_IN_HAL	RETIRED_INST	RETIRED_BR_INST_M	L1_DC_ACCESSES.ALL	IPC	CPI
LU_factor	14330	31226	68	73395	2.18	0.46
SOR_execute	14214	7308	1	14860	0.51	1.94
Random_nextDouble	11943	11194	845	46551	0.94	1.07
SparseCompRow_matmult	10230	28848	4	52557	2.82	0.35
FFT_transform_internal	8831	22071	25	29495	2.50	0.40
MonteCarlo_integrate	6697	6056	453	26205	0.90	1.11
FFT_bitreverse	1317	2845	51	3684	2.16	0.46
Array2D_double_copy	281	547		1387	1.95	0.51
FFT_inverse	227	542	1	1096	2.39	0.42

CPI metric - threshold based performance issue

6.9 ASCII dump of IBS samples

For some usage scenarios, it would be useful to analyze the ascii dump of IBS OP profile samples - perform follow the below mentioned steps:

1. To collect the IBS OP samples, run

```
C:\> AMDuProfCLI.exe collect -e event=ibs-op,interval=100000,loadstore,ibsop-count-
control=1 -a --data-buffer-count 20480 -d 250 -o C:\temp\cpuprof-ibs
```

2. Once the raw file is generated, run the following command to translate and get the ascii dump of IBS OP samples:

```
C:\> AMDuProfCLI.exe translate --ascii event-dump -i C:\temp\cpuprof-ibs.prd
```

3. This will generate the text file that contains ascii dump of the IBS OP samples - `C:\temp\cpuprof-ibs\IbsOpDump.csv`

4. During collection following control knobs are available:
   - *-e event=ibs-op,interval=100000,loadstore,ibsop-count-control=1*
     - interval → sampling interval
     - loadstore → collect only the load & store ops (Windows only option)

- ibsop-count-control=1 → count dispatched micro-ops (0 for "count clock cycles")
- --data-buffer-count 1024 → number of per-core data buffers to allocate

In case if there are too many missing records then try any of the following:

- Increase the sampling interval
- Increase the data buffer count
- Reduce the number of cores profiled

6.10 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 AMD EPYC 1st generation B1 parts, only one PMC register is used at a time for Core PMC event-based profiling (EBP).

Chapter 7 Performance Analysis (Linux only)

This chapter explains the Linux specific performance analysis models and for the common Performance analysis refer *this* chapter.

7.1 OpenMP Analysis

The OpenMP API uses the fork-join model of parallel execution. The program starts with a single master thread to run the serial code and when a parallel region is encountered multiple threads perform the implicit or explicit tasks defined by the OpenMP directives. At the end of that parallel region, the threads join at the barrier and only the master thread continues to execute.

When the threads execute the parallel region code, they should utilize all the available CPU cores and the CPU utilization should be maximized. But due to several reasons the threads wait without doing useful work:

- **Idle**: A thread finishes it task within the parallel region and waits at the barrier for the other threads to complete.
- **Sync**: If locks are used inside the parallel region, threads can wait on synchronization locks to acquire the shared resource.
- **Overhead**: Thread management overhead.

The OpenMP Analysis helps to trace the activities performed by OpenMP threads and their states and provide the thread state timeline for parallel regions to analyze the performance issues.

Support matrix:

Component	Supported Versions	Languages
OpenMP Spec	OpenMP v5.0	
Compiler	LLVM 8, 9, 10, 11	C, C++
	AOCC 2.1, 2.2, 2.3, 3.0	C, C++, Fortran
	ICC 19.1	C, C++, Fortran
OS	Ubuntu 18.04 LTS, 20.04 LTS	
	RHEL 8	
	CentOS 8	

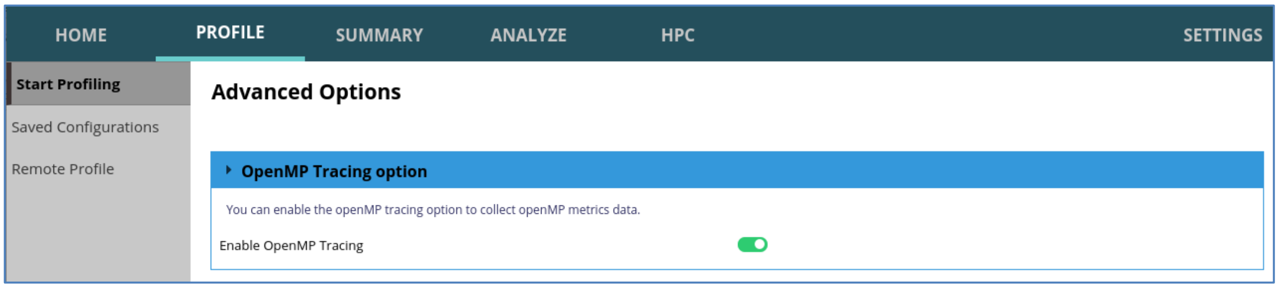
Prerequisites

- Compile the OpenMP application using a supported compiler (on a supported platform) with the required compiler options to enable OpenMP.

7.1.1 Profiling OpenMP Application using GUI

To configure and start profile:

To enable OpenMP profiling, after selecting profile target and profile type, click on **Advanced Options** button to turn on the **Enable OpenMP Tracing** option in **Enable OpenMP Tracing** pane, as seen in the below screen



You can enable the openMP tracing option to collect openMP metrics data.

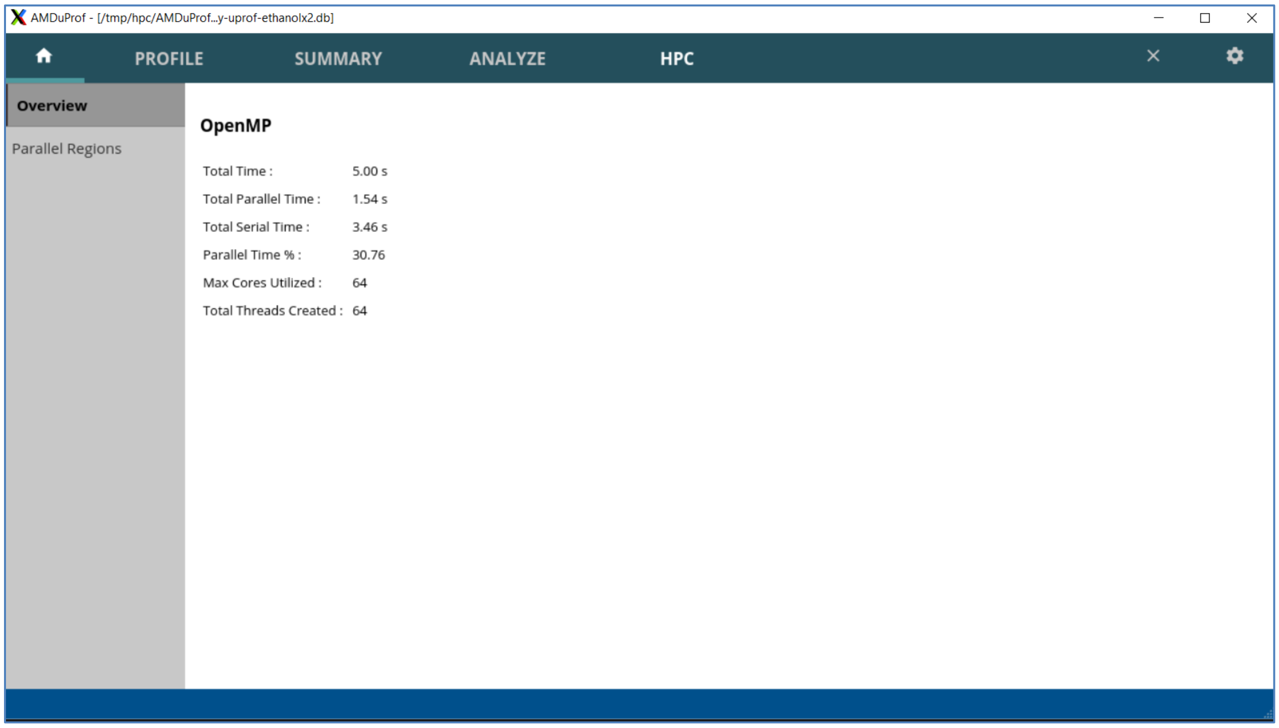
Enable OpenMP Tracing

Advanced Options – Enable OpenMP Tracing

Analyzing the OpenMP report:

After the profile completion, navigate to **HPC** page to analyze the OpenMP tracing data. This page has the following views that can be navigated through the left vertical pane.

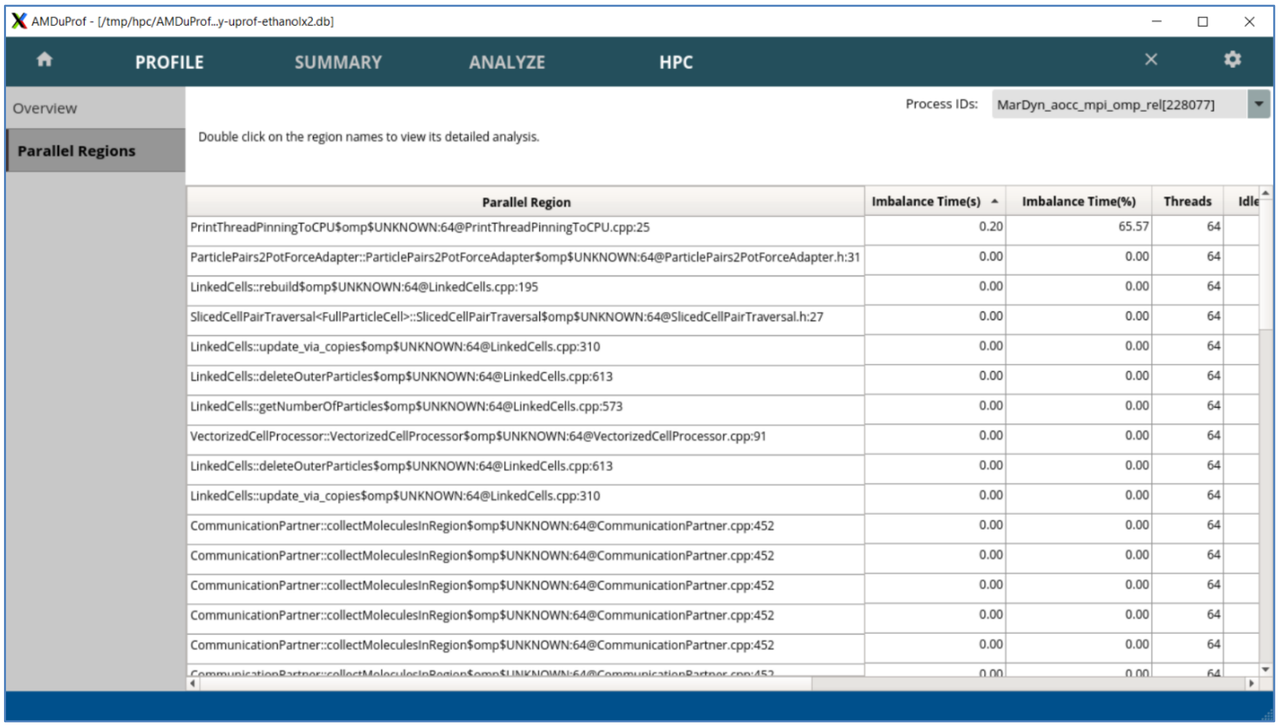
- **Overview** that shows the quick details about the runtime.
- **Parallel Regions** that show the summary of all the parallel regions. This tab is useful to quickly understand which parallel region might be load imbalanced. Double click on the region names to open the 'Regions Detailed Analysis' page.
- **Regions Detailed Analysis** that shows the activity of the threads in a parallel region. If a thread spends too much time on non-work activity, it should be further investigated and optimized to reduce the non-work activity time.



OpenMP

Total Time :	5.00 s
Total Parallel Time :	1.54 s
Total Serial Time :	3.46 s
Parallel Time % :	30.76
Max Cores Utilized :	64
Total Threads Created :	64

HPC – Overview page

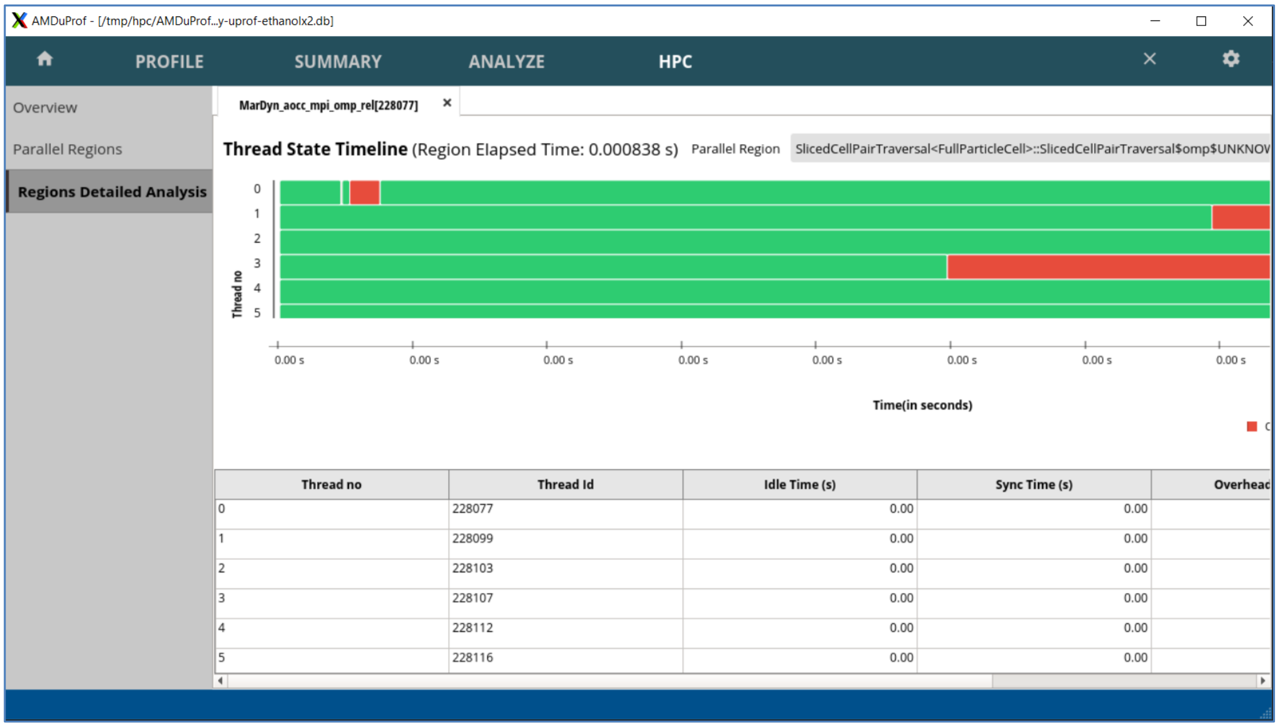


Process IDs: MarDyn_aocc_mpi_omp_rel[228077]

Double click on the region names to view its detailed analysis.

Parallel Region	Imbalance Time(s)	Imbalance Time(%)	Threads	Idle
PrintThreadPinningToCPU$omp$UNKNOWN:64@PrintThreadPinningToCPU.cpp:25	0.20	65.57	64	
ParticlePairs2PotForceAdapter::ParticlePairs2PotForceAdapter$omp$UNKNOWN:64@ParticlePairs2PotForceAdapter.h:31	0.00	0.00	64	
LinkedCells::rebuild$omp$UNKNOWN:64@LinkedCells.cpp:195	0.00	0.00	64	
SlicedCellPairTraversal<FullParticleCell>::SlicedCellPairTraversal$omp$UNKNOWN:64@SlicedCellPairTraversal.h:27	0.00	0.00	64	
LinkedCells::update_via_copies$omp$UNKNOWN:64@LinkedCells.cpp:310	0.00	0.00	64	
LinkedCells::deleteOuterParticles$omp$UNKNOWN:64@LinkedCells.cpp:613	0.00	0.00	64	
LinkedCells::getNumberOfParticles$omp$UNKNOWN:64@LinkedCells.cpp:573	0.00	0.00	64	
VectorizedCellProcessor::VectorizedCellProcessor$omp$UNKNOWN:64@VectorizedCellProcessor.cpp:91	0.00	0.00	64	
LinkedCells::deleteOuterParticles$omp$UNKNOWN:64@LinkedCells.cpp:613	0.00	0.00	64	
LinkedCells::update_via_copies$omp$UNKNOWN:64@LinkedCells.cpp:310	0.00	0.00	64	
CommunicationPartner::collectMoleculesInRegion$omp$UNKNOWN:64@CommunicationPartner.cpp:452	0.00	0.00	64	
CommunicationPartner::collectMoleculesInRegion$omp$UNKNOWN:64@CommunicationPartner.cpp:452	0.00	0.00	64	
CommunicationPartner::collectMoleculesInRegion$omp$UNKNOWN:64@CommunicationPartner.cpp:452	0.00	0.00	64	
CommunicationPartner::collectMoleculesInRegion$omp$UNKNOWN:64@CommunicationPartner.cpp:452	0.00	0.00	64	
CommunicationPartner::collectMoleculesInRegion$omp$UNKNOWN:64@CommunicationPartner.cpp:452	0.00	0.00	64	

HPC – Parallel Regions view


AMDuProf - [/tmp/hpc/AMDuProf...y-uprof-ethanolx2.db]
PROFILE
SUMMARY
ANALYZE
HPC
Overview
Parallel Regions
Regions Detailed Analysis
MarDyn_aocc_mpi_omp_rel[228077]
Thread State Timeline (Region Elapsed Time: 0.000838 s)
Parallel Region
SlicedCellPairTraversal<FullParticleCell>::SlicedCellPairTraversal$omp$UNKNOV
Thread no
0
1
2
3
4
5
0.00 s
Time(in seconds)

Thread no	Thread Id	Idle Time (s)	Sync Time (s)	Overhead
0	228077	0.00	0.00	
1	228099	0.00	0.00	
2	228103	0.00	0.00	
3	228107	0.00	0.00	
4	228112	0.00	0.00	
5	228116	0.00	0.00	

HPC – Regions Detailed Analysis view

7.1.2 Profiling OpenMP Application using CLI

Collect profile data:

Use the following command to profile OpenMP application using uProf CLI

```
$ ./AMDuProfCLI collect --omp --config tbp -o /tmp/myapp_perf <openmp-app>
```

While performing the regular profiling, add option '--omp' to enable OpenMP profiling. This command will launch the program and collect the profile data required to generate the OpenMP analysis report. The data file `/tmp/myapp_perf.caperf` will contain raw profile data.

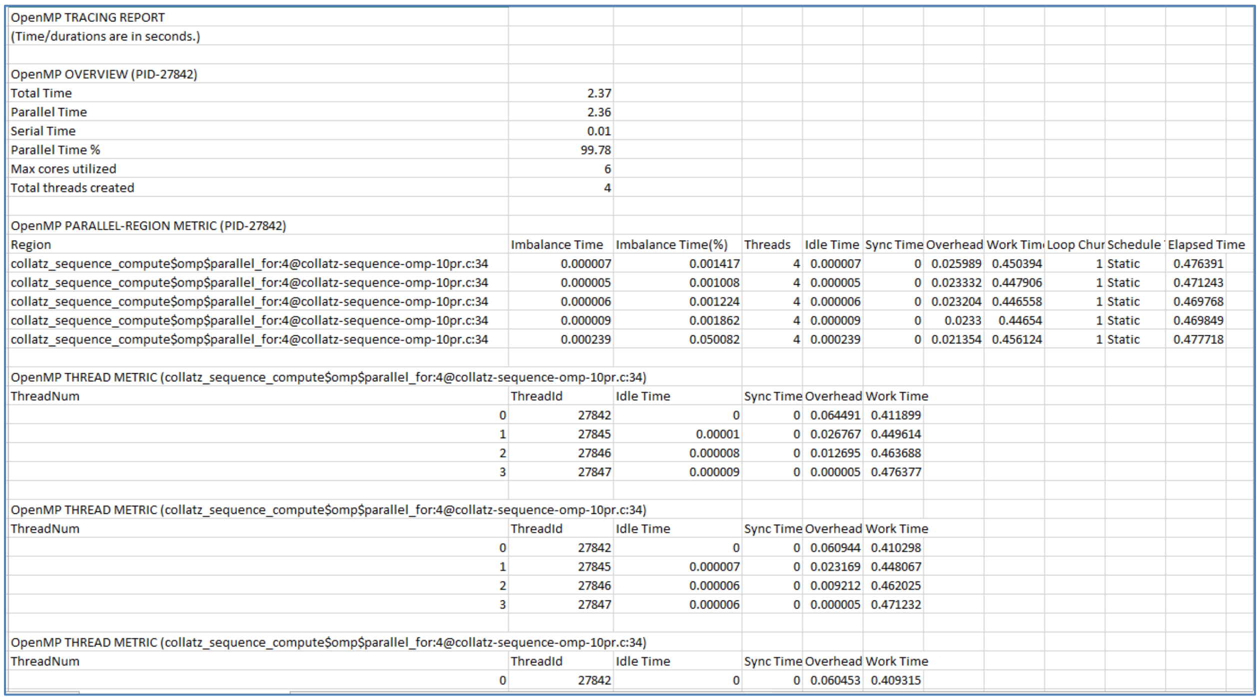
Generate profile report:

Generate CSV report using the `AMDuProfCLI report` command. No additional option needed for OpenMP report generation. uProf checks for availability of any OpenMP profiling data and includes it in the report if available.

```
$ ./AMDuProfCLI report -i /tmp/myapp_perf.caperf
```

This will generate a CSV report at `/tmp/myapp_perf/myapp_perf.csv` and this report will have the following sections:

An example of OpenMP report section in the CSV file shown below.



OpenMP TRACING REPORT										
(Time/durations are in seconds.)										
OpenMP OVERVIEW (PID-27842)										
Total Time	2.37									
Parallel Time	2.36									
Serial Time	0.01									
Parallel Time %	99.78									
Max cores utilized	6									
Total threads created	4									
OpenMP PARALLEL-REGION METRIC (PID-27842)										
Region	Imbalance Time	Imbalance Time(%)	Threads	Idle Time	Sync Time	Overhead	Work Tim	Loop Chur	Schedule	Elapsed Time
collatz_sequence_compute$omp$parallel_for:4@collatz-sequence-omp-10pr.c:34	0.000007	0.001417	4	0.000007	0	0.025989	0.450394	1	Static	0.476391
collatz_sequence_compute$omp$parallel_for:4@collatz-sequence-omp-10pr.c:34	0.000005	0.001008	4	0.000005	0	0.023332	0.447906	1	Static	0.471243
collatz_sequence_compute$omp$parallel_for:4@collatz-sequence-omp-10pr.c:34	0.000006	0.001224	4	0.000006	0	0.023204	0.446558	1	Static	0.469768
collatz_sequence_compute$omp$parallel_for:4@collatz-sequence-omp-10pr.c:34	0.000009	0.001862	4	0.000009	0	0.0233	0.44654	1	Static	0.469849
collatz_sequence_compute$omp$parallel_for:4@collatz-sequence-omp-10pr.c:34	0.000239	0.050082	4	0.000239	0	0.021354	0.456124	1	Static	0.477718
OpenMP THREAD METRIC (collatz_sequence_compute$omp$parallel_for:4@collatz-sequence-omp-10pr.c:34)										
ThreadNum	ThreadId	Idle Time	Sync Time	Overhead	Work Time					
0	27842	0	0	0.064491	0.411899					
1	27845	0.00001	0	0.026767	0.449614					
2	27846	0.000008	0	0.012695	0.463688					
3	27847	0.000009	0	0.000005	0.476377					
OpenMP THREAD METRIC (collatz_sequence_compute$omp$parallel_for:4@collatz-sequence-omp-10pr.c:34)										
ThreadNum	ThreadId	Idle Time	Sync Time	Overhead	Work Time					
0	27842	0	0	0.060944	0.410298					
1	27845	0.000007	0	0.023169	0.448067					
2	27846	0.000006	0	0.009212	0.462025					
3	27847	0.000006	0	0.000005	0.471232					
OpenMP THREAD METRIC (collatz_sequence_compute$omp$parallel_for:4@collatz-sequence-omp-10pr.c:34)										
ThreadNum	ThreadId	Idle Time	Sync Time	Overhead	Work Time					
0	27842	0	0	0.060453	0.409315					

It has following sub-sections:

- **OpenMP Overview**
- **OpenMP PARALLEL-REGION METRIC**: This helps in understanding the imbalanced region, i.e., a region with less total work time with respect to its total time
- **OpenMP THREAD METRIC**: This helps in understanding how each thread spent its time in the parallel region. If a thread spends too much time on non-work activity, then the parallel region should be optimized further to improve the work time of each thread in that region

7.1.3 Environment variables

- **AMDUPROF_MAX_PR_INSTANCES** – Set the max number of unique parallel regions to be traced. The default value is set to 512.
- **AMDUPROF_MAX_PR_INSTANCE_COUNT** – Set the max number of times one unique parallel region to be traced. The default it is set to 512.

7.1.4 Limitations

The following features not yet supported in this release.

- OpenMP profiling with System-wide profiling scope.
- Loop chunk size and Schedule type when these parameters are specified using `schedule` clause. It shows the default values (i.e., ‘1’ & ‘Static’) in this case.
- Nested parallel regions.
- GPU offloading and related constructs.
- Callstack for individual OpenMP threads.
- OpenMP profiling on Windows and FreeBSD platforms.
- Applications with static linkage of OpenMP libraries.

7.2 MPI Profiling

The MPI programs that are launched through mpirun or mpiexec launcher programs can be profiled by uProf. To profile the MPI applications and analyze the data, perform the following the steps:

1. Collect the profile data using CLI collect command.
2. Process the profile data using CLI’s translate command which will generate the profile DB.
3. Import the profile DB in GUI or generate the CSV report using CLI’s report command.

Support matrix:

Component	Supported Versions
MPI Spec	MPI-3.1
MPI Libraries	Open MPI v4.1.0
	MPICH 3.4.1
	ParaStation MPI 5.4.8
	Intel MPI 2019
OS	Ubuntu 18.04 LTS, 20.04 LTS
	RHEL 8
	CentOS 8

7.2.1 Data Collection using CLI

Usually, MPI jobs are launched using MPI launchers like mpirun, mpiexec, etc., We need to use AMDuProfCLI to collect profile data for an MPI application.

MPI job launch using mpirun uses the following syntax:

```
$ mpirun [options] <program> [<args>]
```

AMDuProfCLI is launched using<*program*>and the application is launched using the AMDuProfCLI's arguments. So, profiling an MPI application using AMDuProfCLI uses the following syntax:

```
$ mpirun [options] AMDuProfCLI [options] <program> [<args>]
```

The MPI profiling specific AMDuProfCLI options:

- **`--mpi`** option is to denote that is to profile MPI application. The AMDuProfCLI will collect some additional meta data from MPI processes
- **`--output-dir <output dir>`** specifies the path to a directory in which the profile files are saved. For each MPI process a corresponding raw profile file will be created with the following naming convention "`AMDuProf-<hostname>-<TS>-<PID>.caperf`"

A typical command uses the following syntax:

```
$ mpirun -np <n> /tmp/AMDuProf/bin/AMDuProfCLI collect
--config <config-type> --mpi --output-dir <outpit_dir> [mpi_app]
[<mpi_app_options>]
```

If an MPI application is launched on multiple nodes, AMDuProfCLI will profile all the MPI rank processes running on all the nodes and the user can either analyze the data for processes ran on one/many/all nodes.

Method 1 - Profile all the ranks on single or multiple nodes

To collect profile data for all the ranks running on a single node:

```
$ mpirun -np 16 /tmp/AMDuProf/bin/AMDuProfCLI collect --config tbp
--mpi --output-dir /tmp/myapp-perf myapp.exe
```

To collect profile data for all the ranks in multiple nodes, use -H / --host mpirun options or specify -hostfile <hostfile>

```
$ mpirun -np 16 -H host1,host2 /tmp/AMDuProf/bin/AMDuProfCLI collect
--config tbp --mpi --output-dir /tmp/myapp-perf  myapp.exe
```

Method 2 - Profiling a specific rank(s)

To profile only a specify rank running on a host2:

```
$ export AMDUPROFCLI_CMD=/tmp/AMDuProf/bin/AMDuProfCLI collect --config tbp
--mpi --output-dir /tmp/myapp-perf

$ mpirun -np 4 -host host1 myapp.exe : -host host2 -np 2
  $AMDUPROFCLI_CMD myapp.exe
```

Method 3 – Using MPI config file

The mpirun also takes config file as an input and the AMDuProfCLI can also be used with the config file to profile the MPI application config file (myapp_config):

```
#MPI - myapp config file
-host host1 -n 4 myapp.exe
-host host2 -n 2 /tmp/AMDuProf/bin/AMDuProfCLI collect --config tbp --mpi \
--output-dir /tmp/myapp-perf myapp.exe
```

To run this config to collect data only for the MPI processes running on host2

```
$ mpirun --app myapp_config
```

7.2.2 Analyze the data using CLI

The data collected for MPI processes can either be analyzed using the CSV reported by the AMDuProfCLI's report command.

For CLI, following reporting options are possible

- Generating report for a specific MPI process (using the -i option)

  ```
  $ AMDuProfCLI report \
  -i /tmp/myapp-perf/AMDuProf-<hostname>-<Timestamp>-<PID>.caperf
  ```

- Generating report for all the MPI processes ran on the localhost (ex: host1) in which the MPI launcher was launched (using new option `--input-dir`)

  ```
  $ AMDuProfCLI report --input-dir /tmp/myapp-perf/ --host host1
  ```

  This will create an output dir `/tmp/myapp-perf/AMDuProf-Summary-host1/` and under that dir result files `AMDuProf-Summary-host1.db` and `AMDuProf-Summary-host1.csv`

  Option **`--host`** is not mandatory to create the report file for localhost.

- Generating report for all the MPI processes ran on another host (ex: host2) in which the MPI launcher was not launched

  ```
  $ AMDuProfCLI report --input-dir /tmp/myapp-perf/ --host host2
  ```

  This will create an output dir `/tmp/myapp-perf/AMDuProf-Summary-host2/` and under that dir result files `AMDuProf-Summary-host2.db` and `AMDuProf-Summary-host2.csv`

- Generating report for all the MPI processes ran on all the hosts

  ```
  $ AMDuProfCLI report --input-dir /tmp/myapp-perf/ --host all
  ```

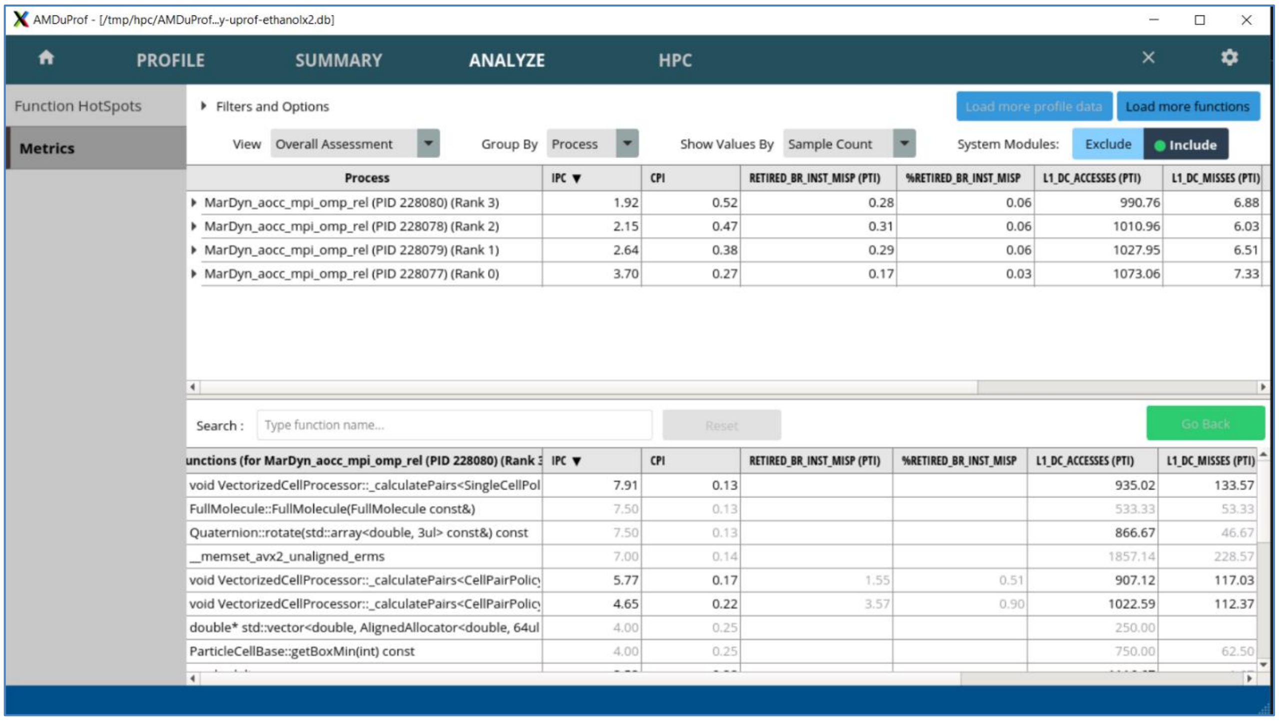
This will create an output dir `/tmp/myapp-perf/AMDuProf-Summary-all/` and under that dir result files `AMDuProf-Summary-all.db` and `AMDuProf-Summary-all.csv`

7.2.3 Analyze the data using GUI

To analyze the profile data in the GUI, run the following steps:

- Generate the profile DB as specified in *this* section
- Import the profile DB as specified in *this* section

After importing, profile data all the profiled ranks will be available for analysis as shown in the below screenshot.



View: Overall Assessment   Group By: Process   Show Values By: Sample Count   System Modules: Exclude / Include

Process	IPC ▼	CPI	RETIRED_BR_INST_MISP (PTI)	%RETIRED_BR_INST_MISP	L1_DC_ACCESSES (PTI)	L1_DC_MISSES (PTI)
MarDyn_aocc_mpi_omp_rel (PID 228080) (Rank 3)	1.92	0.52	0.28	0.06	990.76	6.88
MarDyn_aocc_mpi_omp_rel (PID 228078) (Rank 2)	2.15	0.47	0.31	0.06	1010.96	6.03
MarDyn_aocc_mpi_omp_rel (PID 228079) (Rank 1)	2.64	0.38	0.29	0.06	1027.95	6.51
MarDyn_aocc_mpi_omp_rel (PID 228077) (Rank 0)	3.70	0.27	0.17	0.03	1073.06	7.33

unctions (for MarDyn_aocc_mpi_omp_rel (PID 228080) (Rank 3	IPC ▼	CPI	RETIRED_BR_INST_MISP (PTI)	%RETIRED_BR_INST_MISP	L1_DC_ACCESSES (PTI)	L1_DC_MISSES (PTI)
void VectorizedCellProcessor::_calculatePairs<SingleCellPol	7.91	0.13			935.02	133.57
FullMolecule::FullMolecule(FullMolecule const&)	7.50	0.13			533.33	53.33
Quaternion::rotate(std::array<double, 3ul> const&) const	7.50	0.13			866.67	46.67
__memset_avx2_unaligned_erms	7.00	0.14			1857.14	228.57
void VectorizedCellProcessor::_calculatePairs<CellPairPolicy	5.77	0.17	1.55	0.51	907.12	117.03
void VectorizedCellProcessor::_calculatePairs<CellPairPolicy	4.65	0.22	3.57	0.90	1022.59	112.37
double* std::vector<double, AlignedAllocator<double, 64ul	4.00	0.25			250.00	
ParticleCellBase::getBoxMin(int) const	4.00	0.25			750.00	62.50

MPI – Profile data for all ranks

7.2.4 Limitations

- MPI environment parameters like ‘Total number of ranks’ and ‘Number of ranks running on each node’ are currently supported only for OpenMPI.

7.3 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.

7.4 Profiling Linux Kernel

To profile and analyze the Linux kernel modules and functions, you need to do the following:

1. Enable kernel symbol resolution
2. Download and install kernel debug symbol packages and source

   (or)
3. Build Linux kernel with debug symbols

Once the kernel debug info is available in the default path, uProf automatically locates and utilizes that debug info to show the kernel sources lines and assembly in the source view.

Supported OS	Ubuntu 18.04 LTS, Ubuntu 20.04 LTS, RHEL 7, RHEL 8

7.4.1 Enable kernel symbol resolution

To attribute the kernel samples to appropriate kernel functions, uProf extracts required information from **/proc/kallsyms** file. Exposing kernel symbol addresses through **/proc/kallsyms** requires setting of the appropriate value to **/proc/sys/kernel/kptr_restrict** file.

- Set **/proc/sys/kernel/perf_event_paranoid** config is to -1
- Set **/proc/sys/kernel/kptr_restrict** to appropriate value
  - 0 → kernel addresses are available without limitations
  - 1 → kernel addresses are available if the current user has a CAP_SYSLOG capability
  - 2 → kernel addresses are hidden

Set the perf_event_paranoid value either by

```
$ sudo echo -1 > /proc/sys/kernel/perf_event_paranoid
```

or

```
$ sudo sysctl -w kernel.perf_event_paranoid=-1
```

Set the kptr_restrict value either by

```
$ sudo echo 0 > /proc/sys/kernel/kptr_restrict
```

or

```
$ sudo sysctl -w kernel.kptr_restrict=0
```

7.4.2 Download and install kernel debug symbol packages

On a Linux system, the /boot dir either contains the compressed vmlinuz or uncompressed vmlinux image. These kernel files are stripped and has no symbol and debug information. If there is no debug info AMDuProf will not be able to attribute samples to kernel functions and hence by default uProf cannot report kernel functions.

Some Linux distros provide debug symbol files for their kernel which can be used for profiling purposes.

Ubuntu:

Follow the below mentioned steps to download kernel debug info and source code on Ubuntu systems. Verified on Ubuntu 18.04.03 LTS.

1. Trust the debug symbol signing key

   `$ sudo apt install ubuntu-dbgsym-keyring` // Ubuntu 18.04 LTS and newer:

   `$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys F2EDC64DC5AEE1F6B9C621F0C8CAB6595FDFF622` // Earlier releases of Ubuntu use:

2. Add the debug symbol repository

   ```
   $ echo "deb http://ddebs.ubuntu.com $(lsb_release -cs) main restricted
   universe multiverse
   deb http://ddebs.ubuntu.com $(lsb_release -cs)-security main restricted
   universe multiverse
   deb http://ddebs.ubuntu.com $(lsb_release -cs)-updates main restricted
   universe multiverse
   deb http://ddebs.ubuntu.com $(lsb_release -cs)-proposed main restricted
   universe multiverse" | \
   sudo tee -a /etc/apt/sources.list.d/ddebs.list
   ```

3. Retrieve the list of available debug symbol packages

   ```
   $ sudo apt update
   ```

4. Install the debug symbols for the current kernel version

   ```
   $ sudo apt install --yes linux-image-$(uname -r)-dbgsym
   ```

5. Download the kernel source

   ```
   $ sudo apt source linux-image-unsigned-$(uname -r)
   ```
   or

   ```
   $ sudo apt source linux-image-$(uname -r)
   ```

Once the kernel debug info file gets downloaded, it can be found at the default path:

```
$ /usr/lib/debug/boot/vmlinux-`uname -r`
```

RHEL:

Follow the steps mentioned at the page *https://access.redhat.com/solutions/9907* to download the RHEL kernel debug info.

Once the kernel debug info file gets downloaded, it can be found at the default path:

```
$ /usr/lib/debug/lib/modules/`uname -r`/vmlinux
```

7.4.3 Build Linux kernel with debug symbols

If the debug symbol packages are not available for pre-built kernel images, then to analyze kernel functions at source level requires recompilation of the Linux kernel with debug flag enabled.

7.4.4 How to analyze hotspots in kernel functions:

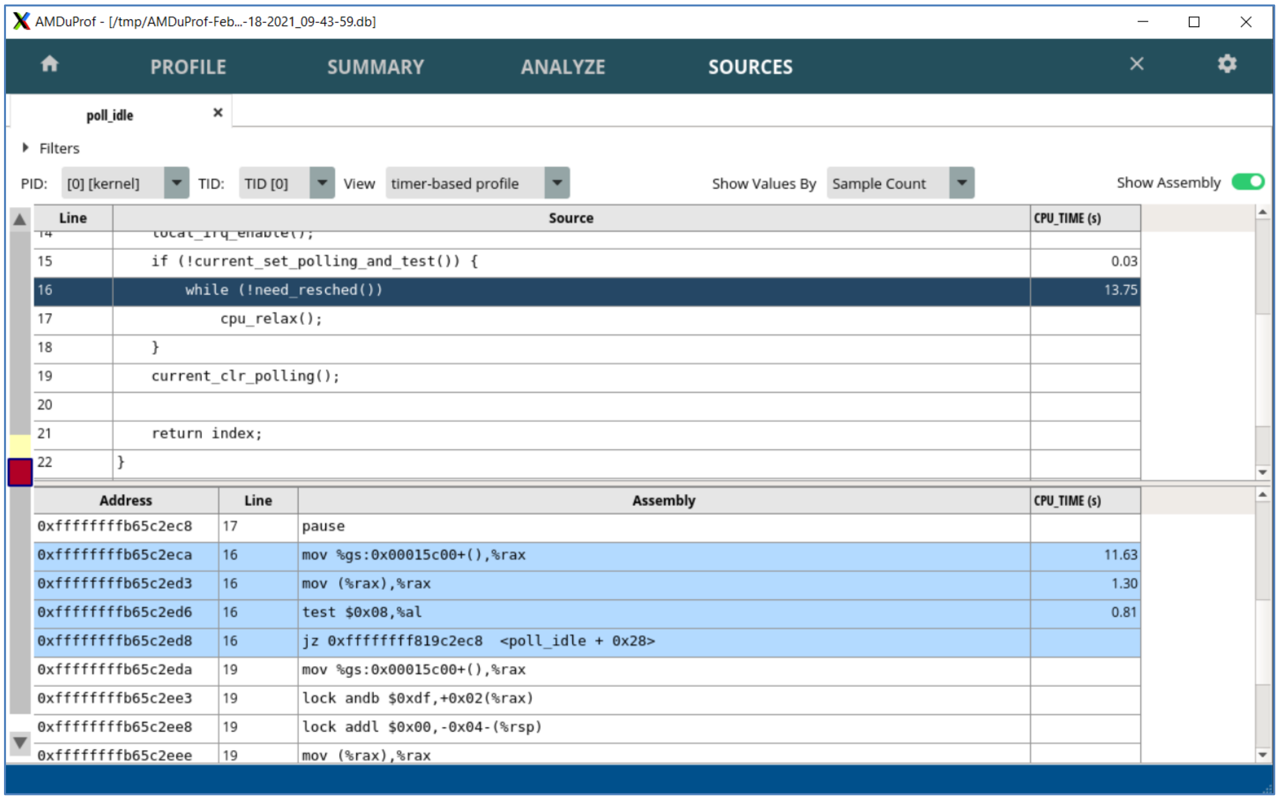
If the debug info for kernel modules is available, any subsequent CPU performance analysis will attribute the kernel space samples appropriately to **[vmlinux]** module and display the hot kernel functions. Otherwise, kernel samples will be attributed to **[kernel.kallsyms]_text** module.

1. If you see **[vmlinux]** module, then you should be able to analyze the performance data for kernel functions in the Source view and IMIX view on GUI. The CLI should also be able to generate source level report and IMIX report for the kernel.

2. If the source is downloaded and the **Source Path** is set while importing the db or in **Sources** section in Advanced Options, then you should be able to see the kernel source lines in GUI.

3. Passing of kernel debug file path, passing of kernel source path is not recommended as that might lead to performance issues.

Below screenshot is the source view of a kernel function.



PID: [0] [kernel] TID: TID [0] View timer-based profile Show Values By Sample Count Show Assembly

Line	Source	CPU_TIME (s)
14	local_irq_enable();	
15	if (!current_set_polling_and_test()) {	0.03
16	while (!need_resched())	13.75
17	cpu_relax();	
18	}	
19	current_clr_polling();	
20		
21	return index;	
22	}	

Address	Line	Assembly	CPU_TIME (s)
0xffffffffb65c2ec8	17	pause	
0xffffffffb65c2eca	16	mov %gs:0x00015c00+(),%rax	11.63
0xffffffffb65c2ed3	16	mov (%rax),%rax	1.30
0xffffffffb65c2ed6	16	test $0x08,%al	0.81
0xffffffffb65c2ed8	16	jz 0xffffffff819c2ec8 <poll_idle + 0x28>	
0xffffffffb65c2eda	19	mov %gs:0x00015c00+(),%rax	
0xffffffffb65c2ee3	19	lock andb $0xdf,+0x02(%rax)	
0xffffffffb65c2ee8	19	lock addl $0x00,-0x04-(%rsp)	
0xffffffffb65c2eee	19	mov (%rax),%rax	

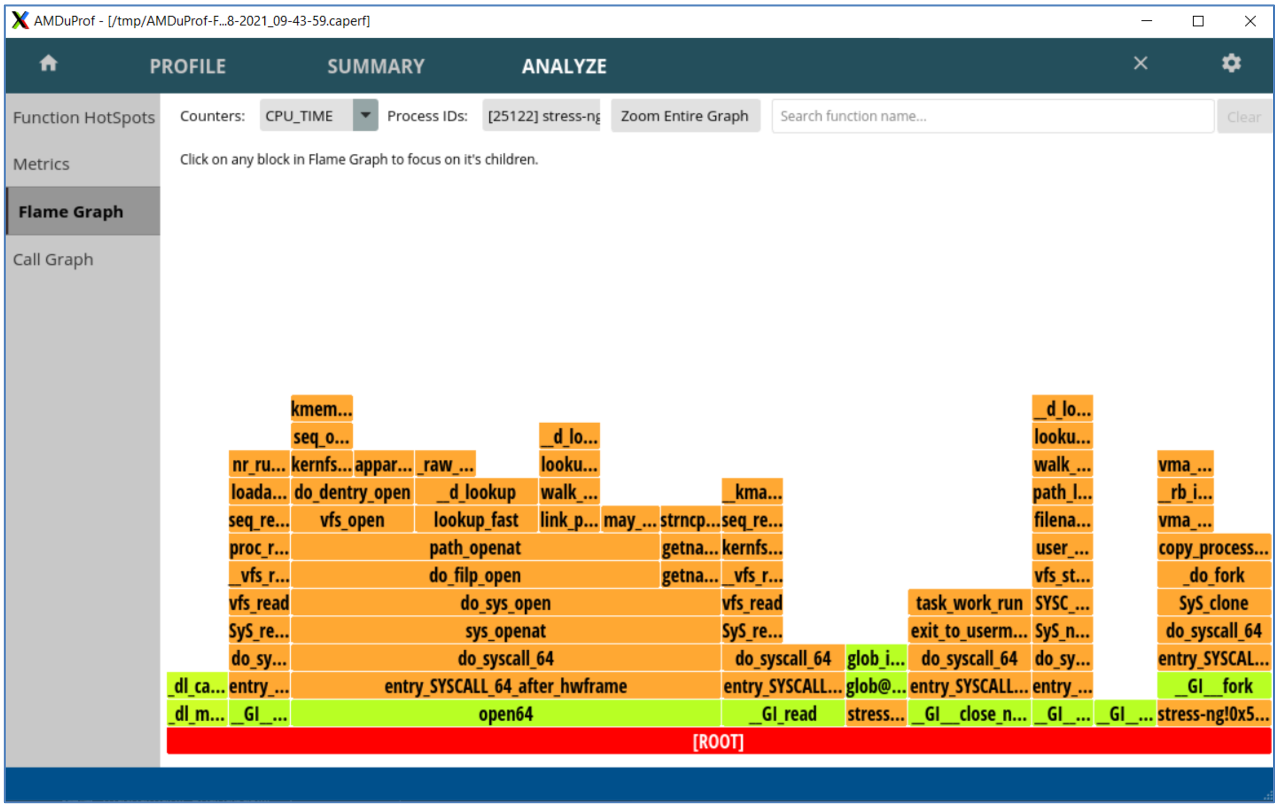
Linux Kernel function – Source view

7.4.5 Linux kernel callstack sampling

In System-wide profile callstack samples too can be collected for kernel functions. For example, the below command will collect the kernel callstack:

```
# AMDuProfCLI collect -a -g /usr/bin/stress-ng --cpu 8 --io 4 --vm 2 --vm-bytes
128M --fork 4 --timeout 20s
```

Below is the screenshot of Flamegraph constructed for the kernel-space callstack samples:



Kernel callstack - Flamegraph

7.4.6 Constraints

1. Do not move the downloaded kernel debug info from its default path.
2. If the kernel version gets upgraded, then download the kernel debug info for the latest kernel version. uProf would fail to show correct source and assembly if there is any mismatch of kernel debug info and kernel version.
3. While profiling or analyzing kernel samples, do not reboot the system in between. Rebooting the system, causes the kernel to load at a different virtual address due to KASLR feature of Linux kernel.
4. 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 source and assembly level analysis, call-graph analysis.

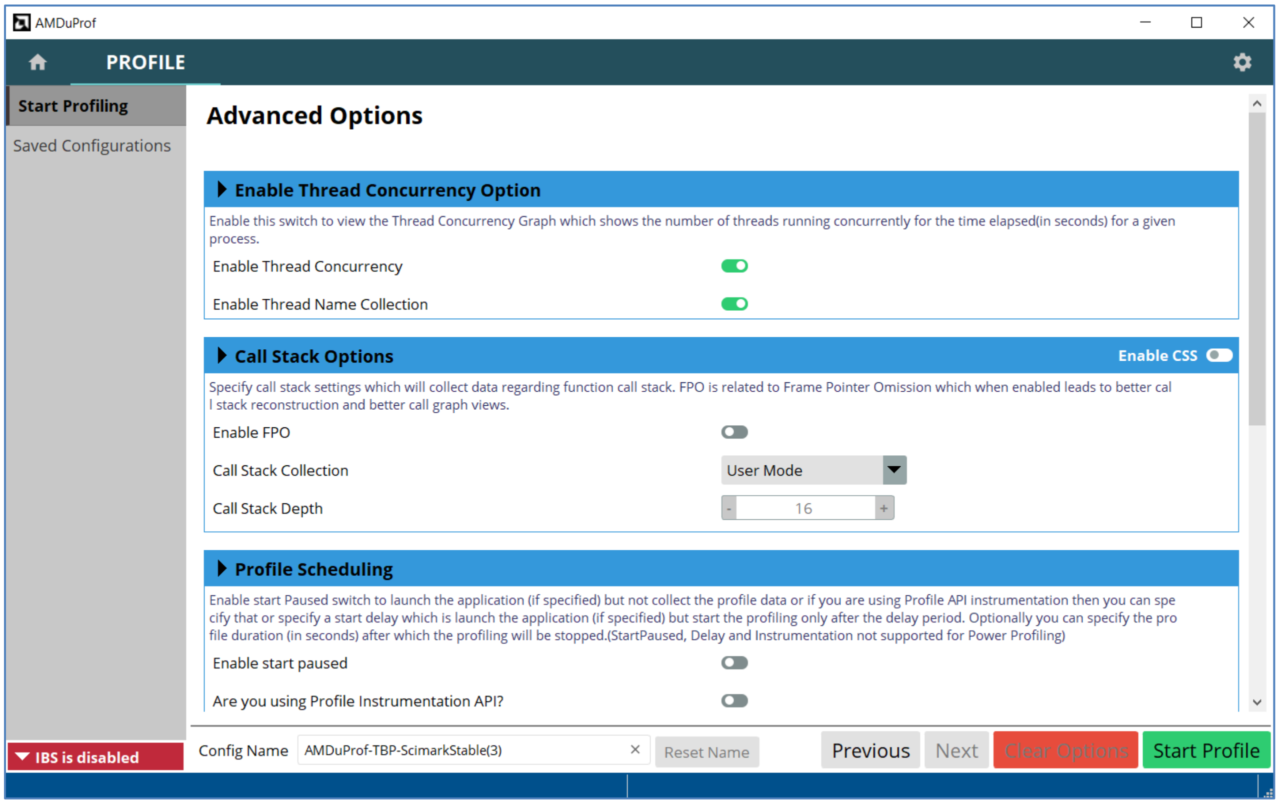
Chapter 8 Performance Analysis (Windows)

8.1 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** switch in **Enable Thread Concurrency Option** pane, as seen in the below screen.



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: on
Enable Thread Name Collection: on

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 cal l stack reconstruction and better call graph views.

Enable FPO: off
Call Stack Collection: User Mode
Call Stack Depth: 16

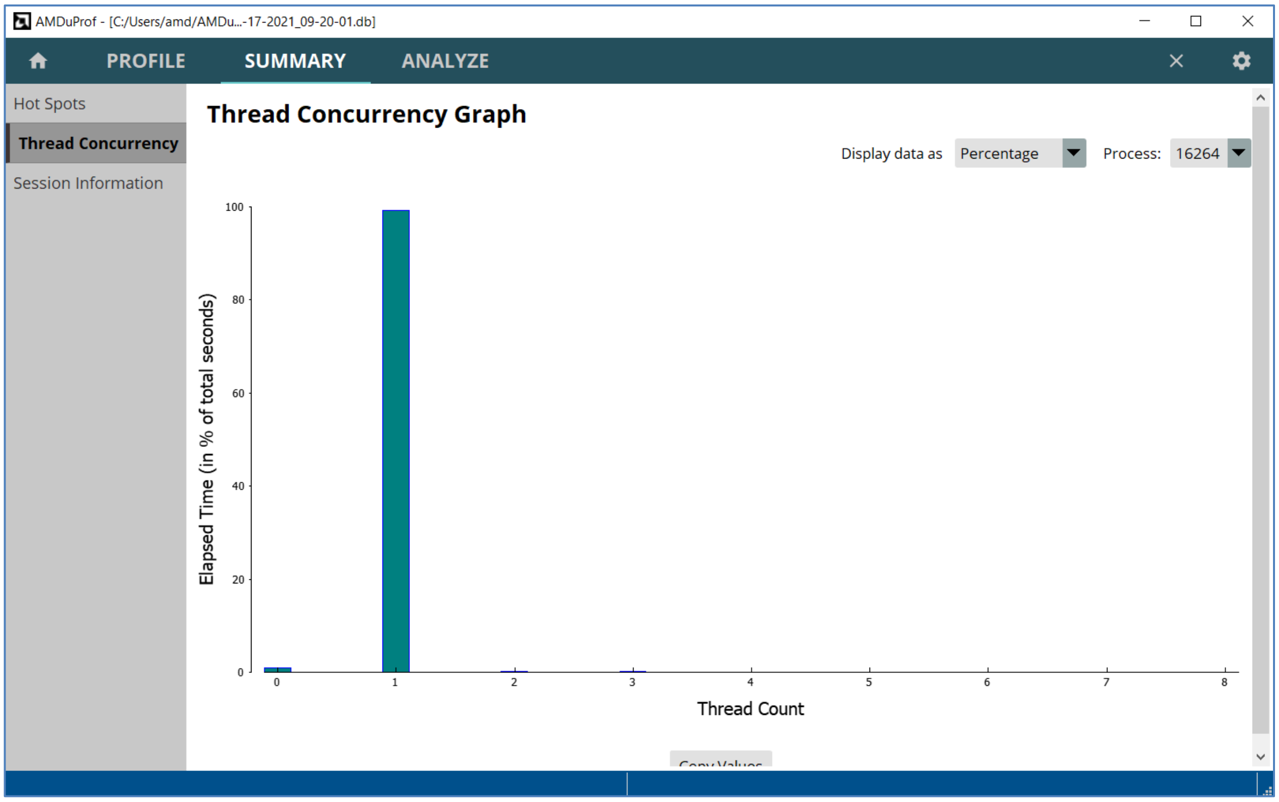
Enable start Paused switch to launch the application (if specified) but not collect the profile data or if you are using Profile API instrumentation then you can spe cify that or specify a start delay which is launch the application (if specified) but start the profiling only after the delay period. Optionally you can specify the pro file duration (in seconds) after which the profiling will be stopped.(StartPaused, Delay and Instrumentation not supported for Power Profiling)

Enable start paused: off
Are you using Profile Instrumentation API?: off

Config Name: AMDuProf-TBP-ScimarkStable(3)

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

Chapter 9 Power Profile

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

9.1 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, EPYC 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 Core and Package.
Frequency	Core Effective Frequency for the sampling period, reported in MHz
Temperature	Average temperature for the sampling period, reported in Celsius. Temperature reported is with reference to Tctl. Available for Package.
P-State	CPU Core P-State at the time when sampling was performed.

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 Core and Package.
Frequency	Core Effective Frequency for the sampling period, reported in MHz
Temperature	Average temperature for the sampling period, reported in Celsius. Temperature reported is with reference to Tctl. Available for Package.
P-State	CPU Core P-State at the time when sampling was performed.

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

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 Core and Package.
Frequency	Core Effective Frequency for the sampling period, reported in MHz
P-State	CPU Core P-State at the time when sampling was performed.

Temperature	Average temperature for the sampling period, reported in Celsius. Temperature reported is with reference to Tctl. Available for Package.

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

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 Core and Package.
Frequency	Core Effective Frequency for the sampling period, reported in MHz
P-State	CPU Core P-State at the time when sampling was performed.
Temperature	Average temperature for the sampling period, reported in Celsius. Temperature reported is with reference to Tctl. Available for Package.

AMD EPYC 3rd generation processors

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 Core and Package.
Frequency	Core Effective Frequency for the sampling period, reported in MHz
P-State	CPU Core P-State at the time when sampling was performed.
Temperature	Average temperature for the sampling period, reported in Celsius. Temperature reported is with reference to Tctl. Available for Package.

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

9.2 Profile using GUI

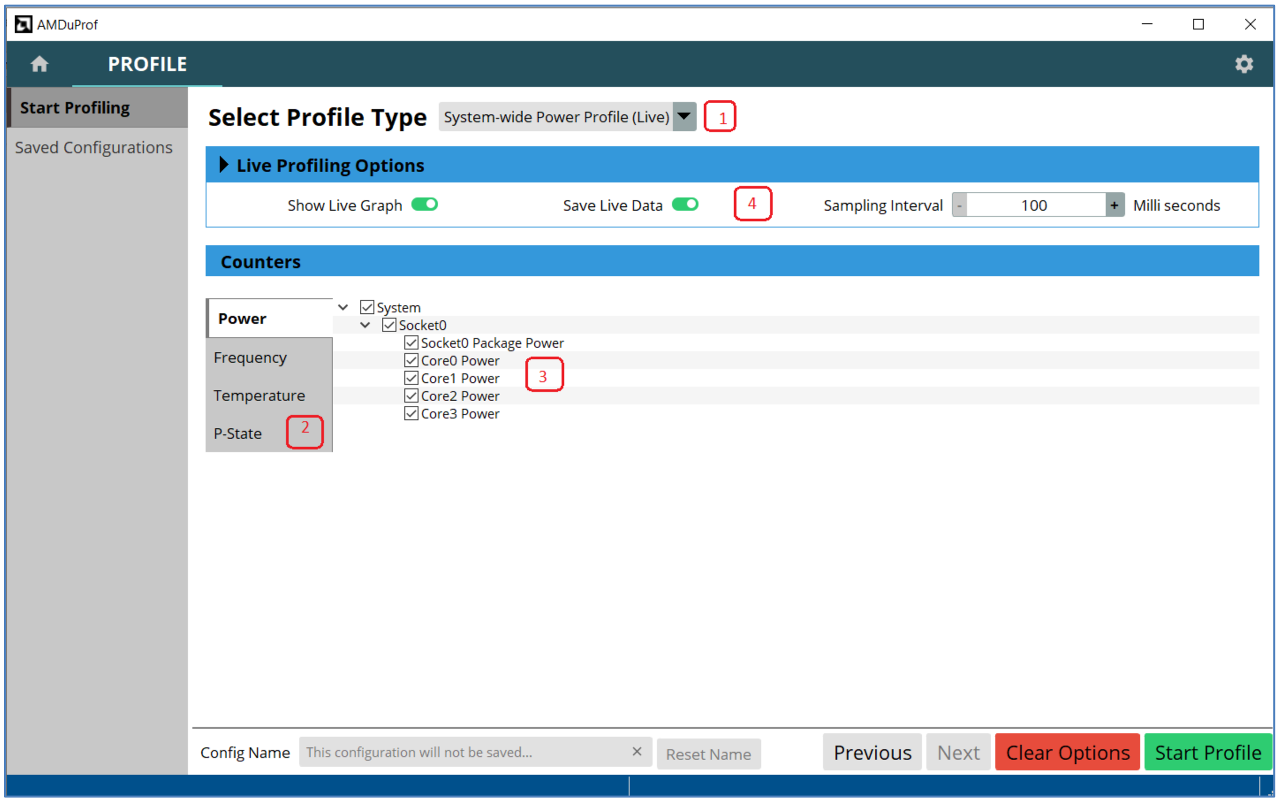
System-wide Power Profile (Live): This profile type is used to perform the power analysis where the metrics are plotted in a live timeline graph and/or saved in a DB. Here are the steps to configure and start the profile:

9.2.1 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.



Start Profiling – Select Profile Type (Live Power Profile)

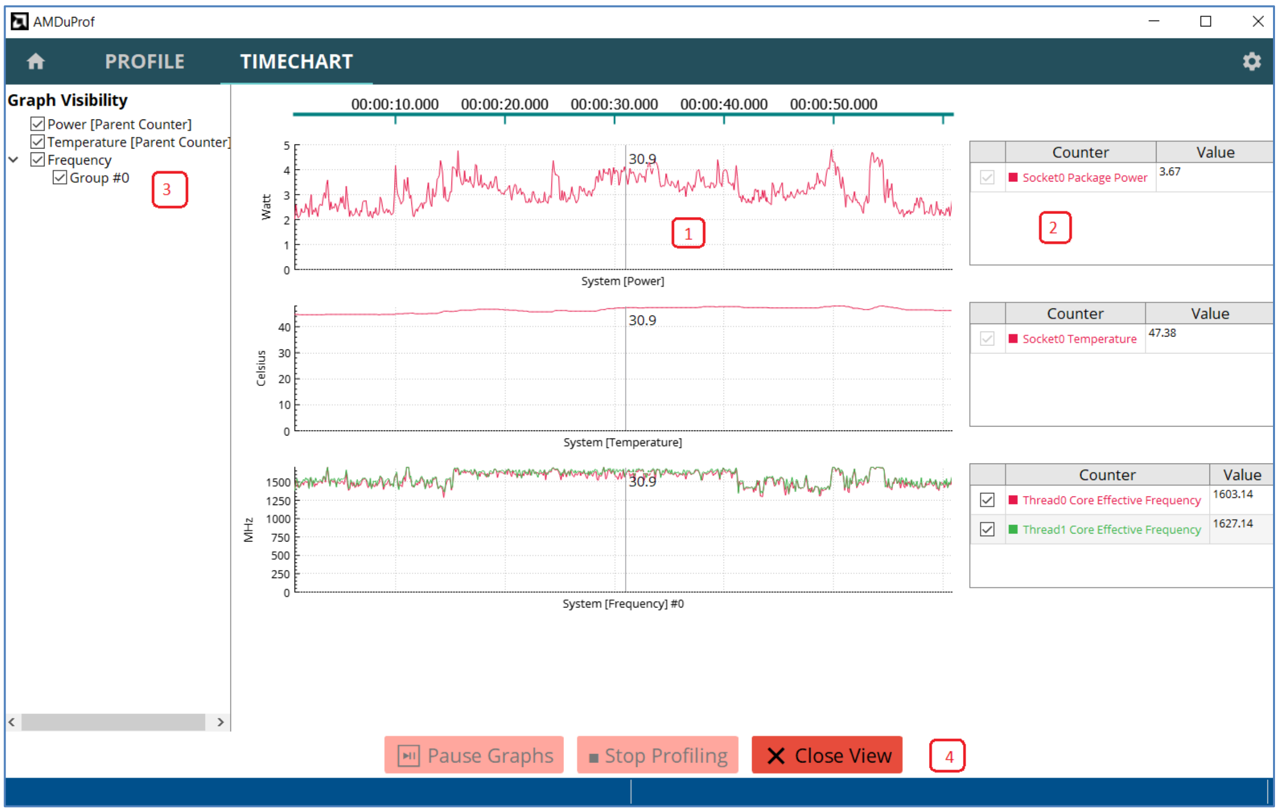
1. Select profile type as **System-wide Power Profile (Live)** from the drop-down list. This will list all the supported counter categories.

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. Options 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.

9.2.2 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 Power	3.67

	Counter	Value
✓	🟥 Socket0 Temperature	47.38

	Counter	Value
✓	🟥 Thread0 Core Effective Frequency	1603.14
✓	🟩 Thread1 Core Effective Frequency	1627.14

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.

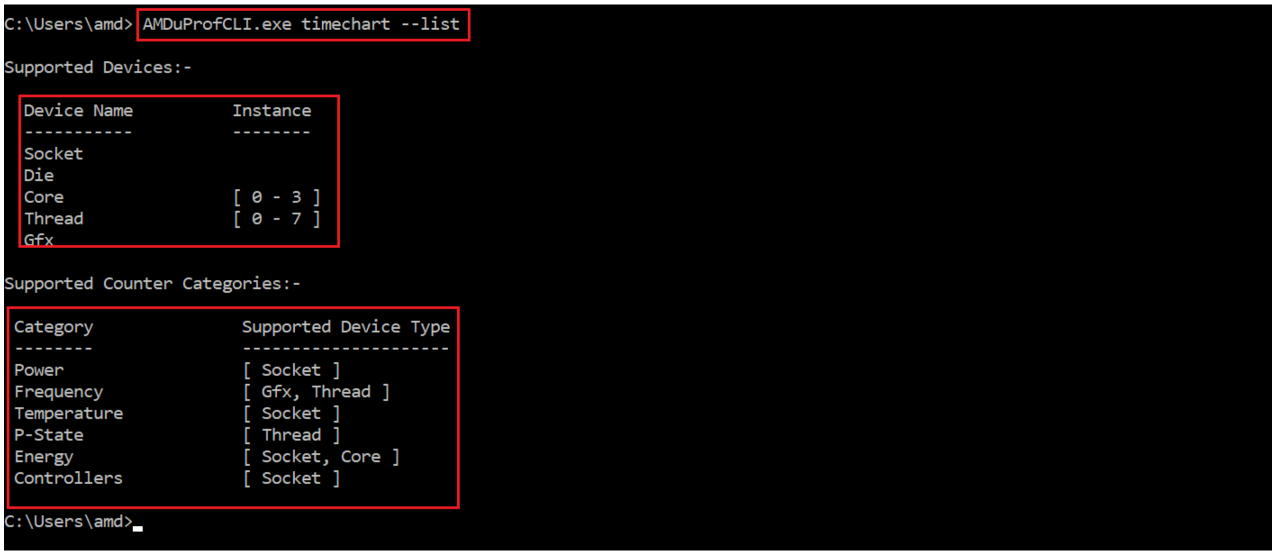
2. There is also a corresponding data table adjacent to each graph to display the current value of the counters.
3. Graph Visibility pane on the left vertical pane will let you choose the graph to display.
4. When plotting is in progress various buttons are available, to let you
   - Pause the graphs 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

9.3 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

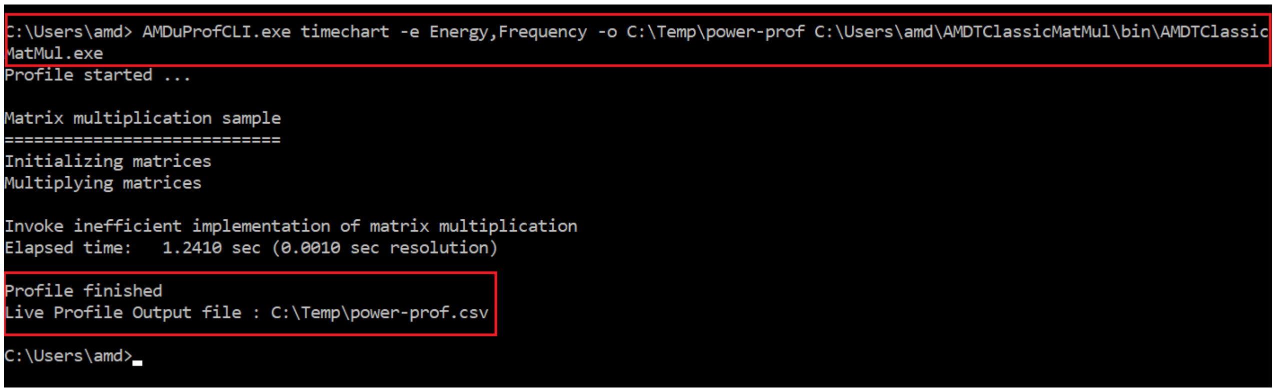
The timechart run to list the supported counter categories:



```
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.

9.3.1 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
  ```

9.4 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

9.4.1 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
```

9.5 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 10 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

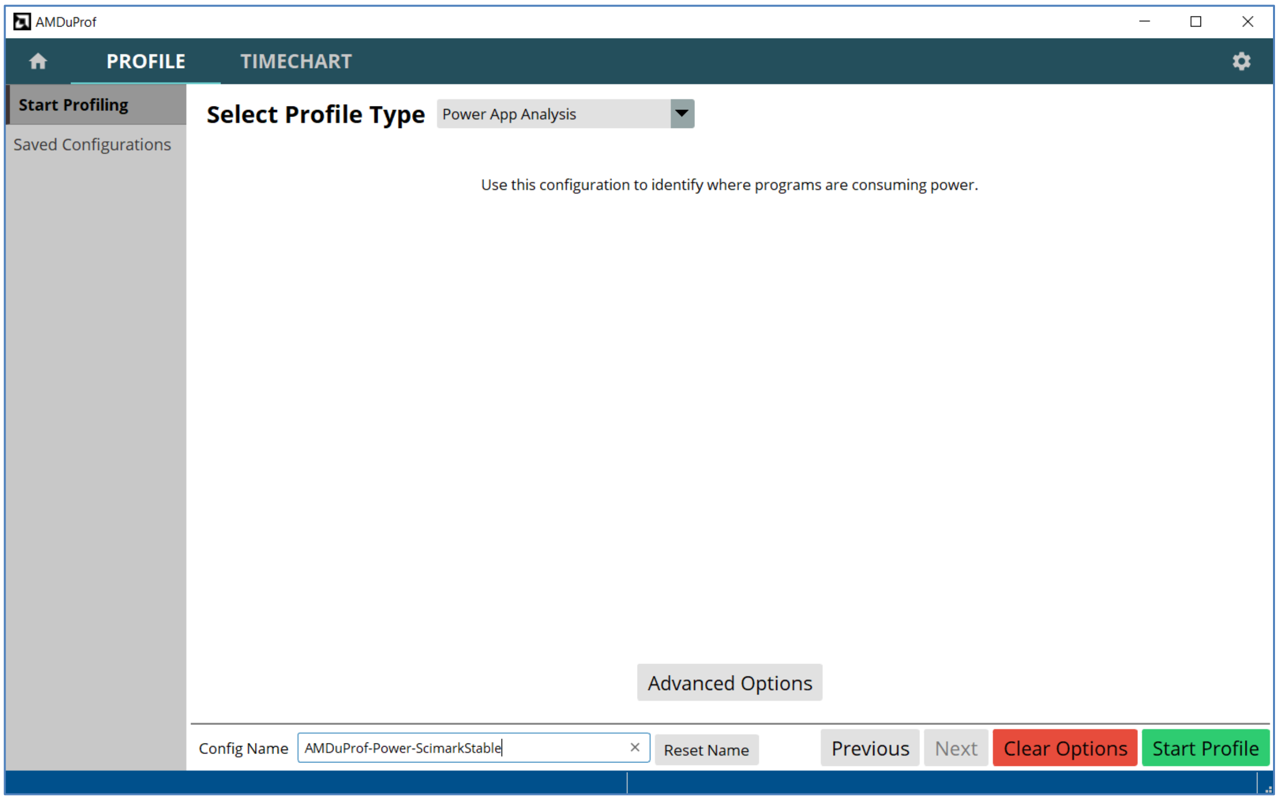
- Profile data
  - Periodically RAPL core energy values are sampled using OS timer as sampling event
- Profile mode
  - Profile data is collected when the application is running in user and kernel mode
- Profiles
  - C, C++, FORTRAN, Assembly applications
  - Various software components – Applications, 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

10.1 Profile using GUI

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

To configure and start profile:

1. Clicking **PROFILE → Start Profiling** will navigate to the **Select Profile Target** window. After selecting the appropriate profile target, clicking **Next** button will take you to **Select Profile Type** fragment.
2. 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

3. Click **Advanced Options** to set symbol paths (if the debug files are in different locations) and other options. Refer *this* section for more information on this window. Callstack is not supported for this profile type.

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. Clicking **ANALYZE** button on the top horizontal navigation bar will go to **Function HotSpots** window. Refer *this* section for more information on this window.

7. Clicking **ANALYZE → Metrics** will display the profile data table at various granularities - Process, Load Modules, Threads and Functions. Refer *this* section for more information on this 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.

10.2 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
```

10.3 Limitations

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

Chapter 11 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 AMDuProfCLI and the data will be collected and processed by the AMDRemoteAgent running in the target system.

Supported configurations:

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

11.1 Profile remote targets using CLI

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

11.1.1 Adding user-id in the remote target system

Before establishing a connection with the remote agent, the user must add the unique UID generated in the host client system. The UID can be generated by using AMDuProfCLI.

To generate unique uid using AMDuProfCLI

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

UID : 10976441267198678299
```

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

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

11.1.2 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...
```

11.1.3 Collect data and generate report

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
```

11.2 Limitations

- Only one instance of CLI client process for a user (having unique client id) can establish connection with AMDRemoteAgent process running on the target system.
- Multiple 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 12 Profile Control APIs

12.1 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
```

12.1.1 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 must 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.

12.1.2 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;
}
```

12.1.3 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++.

12.1.4 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
```

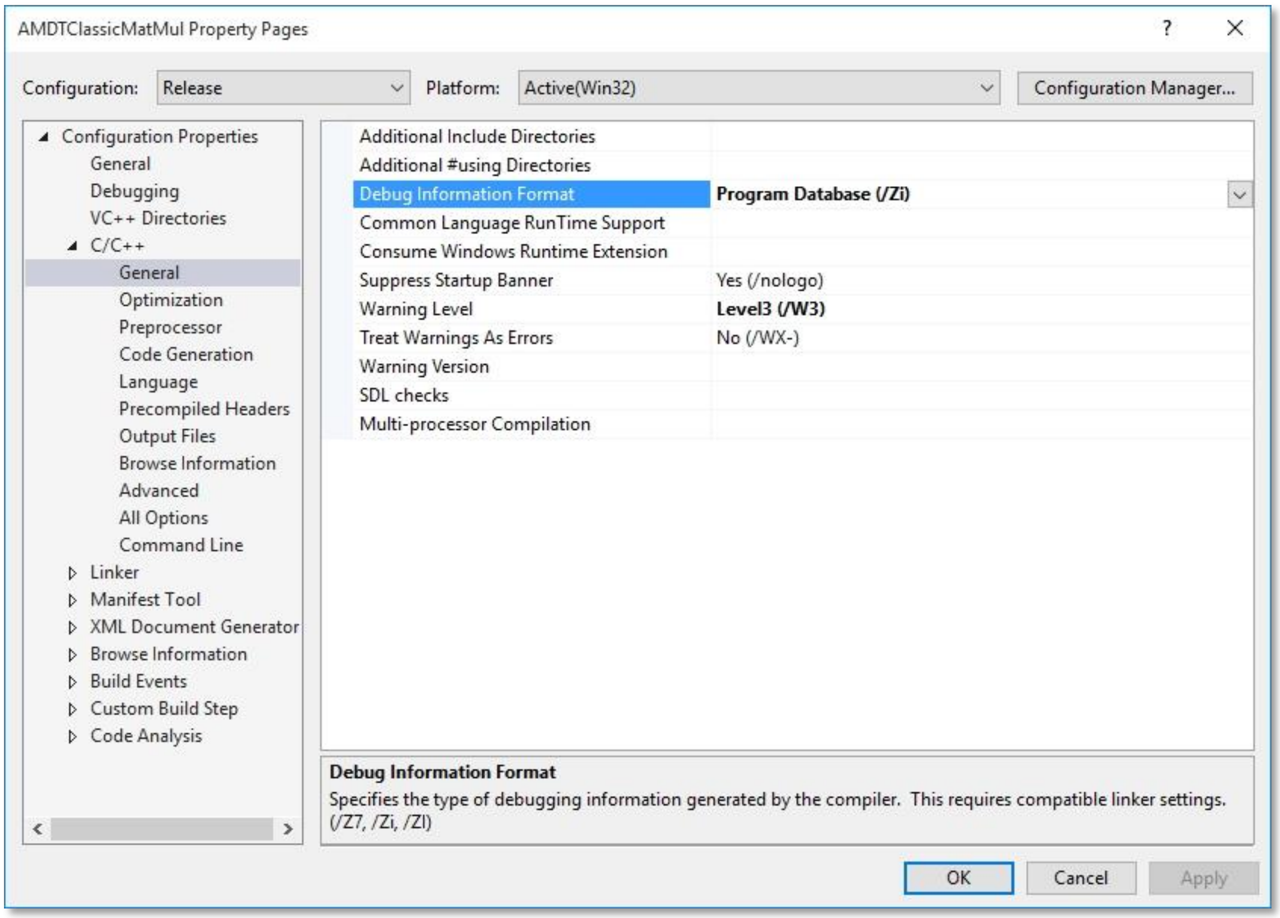
Chapter 13 Reference

13.1 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.

13.1.1 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

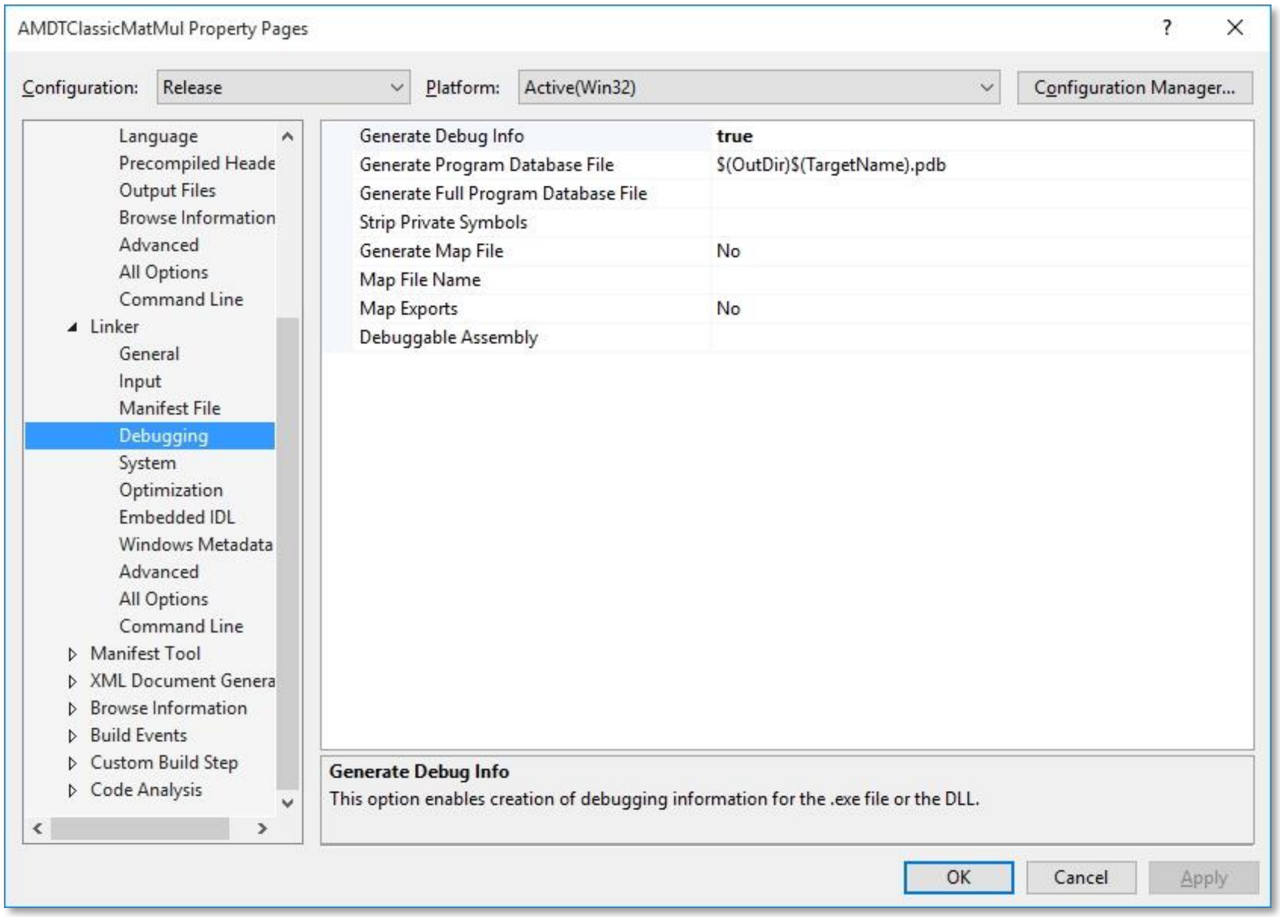
Configuration: Release
Platform: Active(Win32)

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)**.



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).

13.1.2 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.

13.2 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.

13.2.1 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 processor family and model specific PPR for more details.

Data Fabric Performance Monitor Counters (DFPMC)

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

13.2.2 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.

13.2.3 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 an 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.

13.2.4 Predefined Core PMC Events

Some of the interesting Core Performance events of AMD Zen processor models are listed here.

Predefined Core PMC Events – EPYC 2nd generation

Event Id, Unit-mask	Event Abbrev	Name & Description
0x76, 0x00	CYCLES_NOT_IN_HALT	CPU clock cycles not halted The number of cpu cycles when the thread is not in halt state.
0xC0, 0x00	RETIRED_INST	Retired Instructions The number of instructions retired from execution. This count includes exceptions and interrupts. Each exception or interrupt is counted as one instruction.
0xC1, 0x00	RETIRED_MICRO_OPS	Retired Macro Operations The number of macro-ops retired. This count includes all processor activity - instructions, exceptions, interrupts, microcode assists, etc.
0xC2, 0x00	RETIRED_BR_INST	Retired Branch Instructions The number of branch instructions retired. This includes all types of architectural control flow changes, including exceptions and interrupts

0xC3, 0x00	RETIRED_BR_INST_MISP	Retired Branch Instructions Mispredicted The number of retired branch instructions, that were mispredicted. Note that only EX direct mispredicts and indirect target mispredicts are counted.
0x03, 0x08	RETIRED_SSE_AVX_FLOPS	Retired SSE/AVX Flops The number of retired SSE/AVX flops. The number of events logged per cycle can vary from 0 to 64. This is large increment per cycle event, since it can count more than 15 events per cycle. This count both single precision and double precision FP events.
0x29, 0x07	L1_DC_ACCESSES.ALL	All Data cache accesses The number of load and store ops dispatched to LS unit. This counts the dispatch of single op that performs a memory load, dispatch of single op that performs a memory store, dispatch of a single op that performs a load from and store to the same memory address.
0x60, 0x10	L2_CACHE_ACCESS.FROM_L1_IC_MISS	L2 cache access from L1 IC miss The L2 cache access requests due to L1 instruction cache misses.
0x60, 0xC8	L2_CACHE_ACCESS.FROM_L1_DC_MISS	L2 cache access from L1 DC miss The L2 cache access requests due to L1 data cache misses. This also counts hardware and software prefetches.
0x64, 0x01	L2_CACHE_MISS.FROM_L1_IC_MISS	L2 cache miss from L1 IC miss Count all the Instruction cache fill requests that misses in L2 cache
0x64, 0x08	L2_CACHE_MISS.FROM_L1_DC_MISS	L2 cache miss from L1 DC miss Count all the Data cache fill requests that misses in L2 cache
0x71, 0x1F	L2_HWPF_HIT_IN_L3	L2 Prefetcher Hits in L3 Counts all L2 prefetches accepted by the L2 pipeline which miss the L2 cache and hit the L3.
0x72, 0x1F	L2_HWPF_MISS_IN_L2_L3	L2 Prefetcher Misses in L3 Counts all L2 prefetches accepted by the L2 pipeline which miss the L2 and the L3 caches
0x64, 0x06	L2_CACHE_HIT.FROM_L1_IC_MISS	L2 cache hit from L1 IC miss

		Counts all the Instruction cache fill requests that hits in L2 cache.
0x64, 0x70	L2_CACHE_HIT.FROM_L1_DC_MISS	L2 cache hit from L1 DC miss Counts all the Data cache fill requests that hits in L2 cache.
0x70, 0x1F	L2_HWPF_HIT_IN_L2	L2 cache hit from L2 HW Prefetch Counts all L2 prefetches accepted by L2 pipeline which hit in the L2 cache
0x43, 0x01	L1_DEMAND_DC_REFILLS.LOCAL_L2	L1 demand DC fills from L2 The demand Data Cache (DC) fills from local L2 cache to the core.
0x43, 0x02	L1_DEMAND_DC_REFILLS.LOCAL_CACHE	L1 demand DC fills from local CCX The demand Data Cache (DC) fills from same the cache of same CCX or cache of different CCX in the same package (node).
0x43, 0x08	L1_DEMAND_DC_REFILLS.LOCAL_DRAM	L1 demand DC fills from local Memory The demand Data Cache (DC) fills from DRAM or IO connected in the same package (node).
0x43, 0x10	L1_DEMAND_DC_REFILLS.REMOTE_CACHE	L1 demand DC fills from remote cache The demand Data Cache (DC) fills from cache of CCX in the different package (node).
0x43, 0x40	L1_DEMAND_DC_REFILLS.REMOTE_DRAM	L1 demand DC fills from remote Memory The demand Data Cache (DC) fills from DRAM or IO connected in the different package (node).
0x43, 0x5B	L1_DEMAND_DC_REFILLS.ALL	L1 demand DC refills from all data sources. The demand Data Cache (DC) fills from all the data sources.
0x60, 0xFF	L2_REQUESTS.ALL	All L2 cache requests.
0x87, 0x01	STALLED_CYCLES.BACKEND	Instruction pipe stall The Instruction Cache pipeline was stalled during this cycle due to back-pressure.
0x87, 0x02	STALLED_CYCLES.FRONTEND	Instruction pipe stall

		The Instruction Cache pipeline was stalled during this cycle due to upstream queues not providing fetch addresses quickly.
0x84, 0x00	L1_ITLB_MISSES_L2_HITS	L1 TLB miss L2 TLB hit The instruction fetches that misses in the L1 Instruction Translation Lookaside Buffer (ITLB) but hit in the L2-ITLB.
0x85, 0x07	L2_ITLB_MISSES	L1 TLB miss L2 TLB miss The ITLB reloads originating from page table walker. The table walk requests are made for L1-ITLB miss and L2-ITLB misses.
0x45, 0xFF	L1_DTLB_MISSES	L1 DTLB miss The L1 Data Translation Lookaside Buffer (DTLB) misses from load store micro-ops. This event counts both L2-DTLB hit and L2-DTLB miss
0x45, 0xF0	L2_DTLB_MISSES	L1 DTLB miss The L2 Data Translation Lookaside Buffer (DTLB) missed from load store micro-ops
0x47, 0x00	MISALIGNED_LOADS	Misaligned Loads The number of misaligned loads. On Zen3, this event counts the 64B (cache-line crossing) and 4K (page crossing) misaligned loads.
0x52, 0x03	INEFFECTIVE_SW_PF	Ineffective Software Prefetches The number of software prefetches that did not fetch data outside of the processor core. This event counts the Software PREFETCH instruction that saw a match on an already - allocated miss request buffer. Also counts the Software PREFETCH instruction that saw a DC hit.

Predefined Core PMC Events – EPYC 3rd generation

Event Id, Unit-mask	Event Abbrev	Name & Description
0x76, 0x00	CYCLES_NOT_IN_HALT	CPU clock cycles not halted The number of cpu cycles when the thread is not in halt state.
0xC0, 0x00	RETIRED_INST	Retired Instructions

		The number of instructions retired from execution. This count includes exceptions and interrupts. Each exception or interrupt is counted as one instruction.
0xC1, 0x00	RETIRED_MACRO_OPS	Retired Macro Operations The number of macro-ops retired. This count includes all processor activity - instructions, exceptions, interrupts, microcode assists, etc.
0xC2, 0x00	RETIRED_BR_INST	Retired Branch Instructions The number of branch instructions retired. This includes all types of architectural control flow changes, including exceptions and interrupts
0xC3, 0x00	RETIRED_BR_INST_MISP	Retired Branch Instructions Mispredicted The number of retired branch instructions, that were mispredicted. Note that only EX direct mispredicts and indirect target mispredicts are counted.
0x03, 0x08	RETIRED_SSE_AVX_FLOPS	Retired SSE/AVX Flops The number of retired SSE/AVX flops. The number of events logged per cycle can vary from 0 to 64. This is large increment per cycle event, since it can count more than 15 events per cycle. This count both single precision and double precision FP events.
0x29, 0x07	L1_DC_ACCESSES.ALL	All Data cache accesses The number of load and store ops dispatched to LS unit. This counts the dispatch of single op that performs a memory load, dispatch of single op that performs a memory store, dispatch of a single op that performs a load from and store to the same memory address.
0x60, 0x10	L2_CACHE_ACCESS.FROM_L1_IC_MISS	L2 cache access from L1 IC miss The L2 cache access requests due to L1 instruction cache misses.
0x60, 0xE8	L2_CACHE_ACCESS.FROM_L1_DC_MISS	L2 cache access from L1 DC miss The L2 cache access requests due to L1 data cache misses. This also counts hardware and software prefetches.
0x64, 0x01	L2_CACHE_MISS.FROM_L1_IC_MISS	L2 cache miss from L1 IC miss Count all the Instruction cache fill requests that misses in L2 cache

[AMD Public Use]

0x64, 0x08	L2_CACHE_MISS.FROM_L1_DC_MISS	L2 cache miss from L1 DC miss Count all the Data cache fill requests that misses in L2 cache
0x71, 0xFF	L2_HWPF_HIT_IN_L3	L2 Prefetcher Hits in L3 Counts all L2 prefetches accepted by the L2 pipeline which miss the L2 cache and hit the L3.
0x72, 0xFF	L2_HWPF_MISS_IN_L2_L3	L2 Prefetcher Misses in L3 Counts all L2 prefetches accepted by the L2 pipeline which miss the L2 and the L3 caches
0x64, 0x06	L2_CACHE_HIT.FROM_L1_IC_MISS	L2 cache hit from L1 IC miss Counts all the Instruction cache fill requests that hits in L2 cache.
0x64, 0xF0	L2_CACHE_HIT.FROM_L1_DC_MISS	L2 cache hit from L1 DC miss Counts all the Data cache fill requests that hits in L2 cache.
0x70, 0xFF	L2_HWPF_HIT_IN_L2	L2 cache hit from L2 HW Prefetch Counts all L2 prefetches accepted by L2 pipeline which hit in the L2 cache
0x43, 0x01	L1_DEMAND_DC_REFILLS.LOCAL_L2	L1 demand DC fills from L2 The demand Data Cache (DC) fills from local L2 cache to the core.
0x43, 0x02	L1_DEMAND_DC_REFILLS.LOCAL_CACHE	L1 demand DC fills from local CCX The demand Data Cache (DC) fills from the L3 cache or L2 in the same CCX.
0x43, 0x04	L1_DC_REFILLS.EXTERNAL_CACHE_LOCAL	L1 DC fills from local external CCX caches The Data Cache (DC) fills from cache of different CCX in the same package (node).
0x43, 0x08	L1_DEMAND_DC_REFILLS.LOCAL_DRAM	L1 demand DC fills from local Memory The demand Data Cache (DC) fills from DRAM or IO connected in the same package (node).
0x43, 0x10	L1_DEMAND_DC_REFILLS.EXTERNAL_CACHE_REMOTE	L1 demand DC fills from remote external cache The demand Data Cache (DC) fills from cache of CCX in the different package (node).

0x43, 0x40	L1_DEMAND_DC_REFILLS.REMOTE_DRAM	L1 demand DC fills from remote Memory The demand Data Cache (DC) fills from DRAM or IO connected in the different package (node).
0x43, 0x14	L1_DEMAND_DC_REFILLS.EXTENAL_CACHE	L1 demand DC fills from external caches The demand Data Cache (DC) fills from cache of different CCX in the same or different package (node).
0x43, 0x5F	L1_DEMAND_DC_REFILLS.ALL	L1 demand DC refills from all data sources. The demand Data Cache (DC) fills from all the data sources.
0x44, 0x01	L1_DC_REFILLS.LOCAL_L2	L1 DC fills from local L2 The Data Cache (DC) fills from local L2 cache to the core.
0x44, 0x02	L1_DC_REFILLS.LOCAL_CACHE	L1 DC fills from local CCX cache The Data Cache (DC) fills from different L2 cache in the same CCX or L3 cache that belongs to the same CCX.
0x44, 0x08	L1_DC_REFILLS.LOCAL_DRAM	L1 DC fills from local Memory The Data Cache (DC) fills from DRAM or IO connected in the same package (node).
0x44, 0x04	L1_DC_REFILLS.EXTERNAL_CACHE_LOCAL	L1 DC fills from local external CCX caches The Data Cache (DC) fills from cache of different CCX in the same package (node).
0x44, 0x10	L1_DC_REFILLS.EXTERNAL_CACHE_REMOTE	L1 DC fills from remote external CCX caches The Data Cache (DC) fills from cache of CCX in the different package (node).
0x44, 0x40	L1_DC_REFILLS.REMOTE_DRAM	L1 DC fills from remote Memory The Data Cache (DC) fills from DRAM or IO connected in the different package (node).
0x44, 0x14	L1_DC_REFILLS.EXTENAL_CACHE	L1 DC fills from local external CCX caches The Data Cache (DC) fills from cache of different CCX in the same or different package (node).
0x44, 0x48	L1_DC_REFILLS.DRAM	L1 DC fills from local Memory The Data Cache (DC) fills from DRAM or IO connected in the same or different package (node).

0x44, 0x50	L1_DC_REFILLS.REMOTE_NODE	L1 DC fills from remote node The Data Cache (DC) fills from cache of CCX in the different package (node) or the DRAM / IO connected in the different package (node).
0x44, 0x03	L1_DC_REFILLS.LOCAL_CACHE_L2_L3	L1 DC fills from same CCX The Data Cache (DC) fills from local L2 cache to the core or different L2 cache in the same CCX or L3 cache that belongs to the same CCX
0x44, 0x5F	L1_DC_REFILLS.ALL	L1 DC fills from all the data sources The Data Cache fills from all the data sources
0x60, 0xFF	L2_REQUESTS.ALL	All L2 cache requests.
0x87, 0x01	STALLED_CYCLES.BACKEND	Instruction pipe stall The Instruction Cache pipeline was stalled during this cycle due to back-pressure.
0x87, 0x02	STALLED_CYCLES.FRONTEND	Instruction pipe stall The Instruction Cache pipeline was stalled during this cycle due to upstream queues not providing fetch addresses quickly.
0x84, 0x00	L1_ITLB_MISSES_L2_HITS	L1 TLB miss L2 TLB hit The instruction fetches that misses in the L1 Instruction Translation Lookaside Buffer (ITLB) but hit in the L2-ITLB.
0x85, 0x07	L2_ITLB_MISSES	L1 TLB miss L2 TLB miss The ITLB reloads originating from page table walker. The table walk requests are made for L1-ITLB miss and L2-ITLB misses.
0x45, 0xFF	L1_DTLB_MISSES	L1 DTLB miss The L1 Data Translation Lookaside Buffer (DTLB) misses from load store micro-ops. This event counts both L2-DTLB hit and L2-DTLB miss
0x45, 0xF0	L2_DTLB_MISSES	L1 DTLB miss The L2 Data Translation Lookaside Buffer (DTLB) missed from load store micro-ops
0x78, 0xFF	ALL_TLB_FLUSHES	All TLB flushes

0x47, 0x03	MISALIGNED_LOADS	Misaligned Loads The number of misaligned loads. On Zen3, this event counts the 64B (cache-line crossing) and 4K (page crossing) misaligned loads.
0x52, 0x03	INEFFECTIVE_SW_PF	Ineffective Software Prefetches The number of software prefetches that did not fetch data outside of the processor core. This event counts the Software PREFETCH instruction that saw a match on an already - allocated miss request buffer. Also counts the Software PREFETCH instruction that saw a DC hit.

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.
L1_DC_REFILLS.ALL (PTI)	The number of demand data cache (DC) fills per thousand retired instructions. These demand DC fills are from all the data sources like Local L2/L3 cache, remote caches, local memory, and remote memory.
L1_DC_MISSES (PTI)	The number of L2 cache access requests due to L1 data cache misses, per thousand retired instructions. This L2 cache access requests also includes the hardware and software prefetches.

L1_DC_ACCESS_RATE	The DC access rate is the number of DC accesses divided by the total number of retired instructions
L1_DC_MISS_RATE	The DC miss rate is the number of DC misses divided by the total number of retired instructions.
L1_DC_MISS_RATIO	The DC miss ratio is the number of DC misses divided by the total number of DC accesses.

13.2.5 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.

13.3 Useful links

For the processor specific PMC events and their descriptions, refer AMD developer documents.

Processor Programming Reference (PPR) for AMD Family 17h Processors:
https://developer.amd.com/resources/developer-guides-manuals/

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

Software Optimization Guide for AMD Family 19h Processors:
https://www.amd.com/system/files/TechDocs/56665.zip