

AMD

uProf User Guide

Publication # **57368**

Revision # **4.0**

Issue Date **November 2022**

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

Trademarks

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

Dolby is a trademark of Dolby Laboratories.

ENERGY STAR is a registered trademark of the U.S. Environmental Protection Agency.

HDMI is a trademark of HDMI Licensing, LLC.

HyperTransport is a licensed trademark of the HyperTransport Technology Consortium.

Microsoft, Windows, Windows Vista, and DirectX are registered trademarks of Microsoft Corporation.

MMX is a trademark of Intel Corporation.

OpenCL is a trademark of Apple Inc. used by permission by Khronos.

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

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

Dolby Laboratories, Inc.

Manufactured under license from Dolby Laboratories.

Rovi Corporation

This device is protected by U.S. patents and other intellectual property rights. The use of Rovi Corporation's copy protection technology in the device must be authorized by Rovi Corporation and is intended for home and other limited pay-per-view uses only, unless otherwise authorized in writing by Rovi Corporation.

Reverse engineering or disassembly is prohibited.

USE OF THIS PRODUCT IN ANY MANNER THAT COMPLIES WITH THE MPEG-2 STANDARD IS EXPRESSLY PROHIBITED WITHOUT A LICENSE UNDER APPLICABLE PATENTS IN THE MPEG-2 PATENT PORTFOLIO, WHICH LICENSE IS AVAILABLE FROM MPEG LA, L.L.C., 6312 S. FIDDLERS GREEN CIRCLE, SUITE 400E, GREENWOOD VILLAGE, COLORADO 80111.

Contents

Revision History	16
About this Document	17
Intended Audience	17
Conventions	17
Abbreviations	17
Terminology	18
Chapter 1 Introduction	20
1.1 Overview	20
1.2 Specification	21
1.2.1 Processors	21
1.2.2 Operating Systems	21
1.2.3 Compilers and Application Environment	21
1.2.4 Virtualization Support	22
1.2.5 Container Support (Beta Support)	22
1.3 Installing AMD uProf	22
1.3.1 Windows	23
1.3.2 Linux	23
1.3.3 FreeBSD	25
1.4 Sample Programs	25
1.5 Support	25
Chapter 2 Workflow and Key Concepts	26
2.1 Workflow	26
2.1.1 Collect Phase	26
2.1.2 Translate Phase	28
2.1.3 Analyze Phase	28
2.2 Predefined Sampling Configuration	28
2.3 Predefined View Configuration	29
Chapter 3 Getting started with AMDuProfPcm – System Analysis	34
3.1 Overview	34

3.1.1	Prerequisite(s)	34
3.2	Options	35
3.3	Examples	47
3.3.1	Linux and FreeBSD	47
3.3.2	Windows	48
3.4	BIOS Settings - Known Behavior	50
3.5	Pre-release Features	51
3.5.1	Monitoring without Root Privileges	51
3.5.2	Roofline Model	52
3.5.3	Pipeline Utilization (Top-down Analysis)	54
Chapter 4	Getting Started with AMDuProfSys – System Analysis	56
4.1	Overview	56
4.2	Options	58
4.2.1	Generic	58
4.2.2	Collect Command	58
4.2.3	Report Command	59
4.3	Examples	60
Chapter 5	Getting Started with AMD uProf GUI	62
5.1	User Interface	62
5.2	Launching GUI	63
5.3	Configure a Profile	64
5.3.1	Select Profile Target	64
5.3.2	Select Profile Type	65
5.3.3	Advanced Options	67
5.3.4	Start Profile	69
5.4	Analyze the Profile Data	69
5.4.1	Overview of Performance Hotspots	70
5.4.2	Thread Concurrency Graph	71
5.4.3	Function HotSpots	71
5.4.4	Process and Functions	73
5.4.5	Source and Assembly	75

5.4.6	Flame Graph	76
5.4.7	Call Graph	78
5.4.8	IMIX View	79
5.5	Importing Profile Database	80
5.6	Analyzing Saved Profile Session	81
5.7	Using Saved Profile Configuration	82
5.8	Settings	83
Chapter 6	Getting Started with AMD uProf CLI	85
6.1	Overview	85
6.2	Starting a CPU Profile	85
6.2.1	List of Predefined Sample Configurations	86
6.2.2	Profile Report	87
6.3	Starting a Power Profile	88
6.3.1	System-wide Power Profiling (Live)	88
6.4	Collect Command	89
6.4.1	Options	89
6.4.2	Windows Specific Options	92
6.4.3	Linux Specific Options	93
6.4.4	Examples	95
6.5	Report Command	98
6.5.1	Options	99
6.5.2	Windows Specific Options	101
6.5.3	Linux Specific Options	102
6.5.4	Examples	102
6.6	Translate Command	104
6.6.1	Options	104
6.6.2	Windows Specific Options	105
6.6.3	Linux Specific Options	105
6.6.4	Examples	106
6.7	Timechart Command	106
6.7.1	Options	107

6.7.2	Examples	107
6.8	Info Command	108
6.8.1	Options	109
6.8.2	Examples	110
Chapter 7	Performance Analysis	111
7.1	CPU Profiling	111
7.2	Analysis with Time-based Profiling	112
7.2.1	Configuring and Starting Profile	112
7.2.2	Analyzing Profile Data	113
7.3	Analysis with Event-based Profiling	114
7.3.1	Configuring and Starting Profile	114
7.3.2	Analyzing Profile Data	115
7.4	Analysis with Instruction-based Sampling	115
7.4.1	Configuring and Starting Profile	115
7.4.2	Analyzing Profile Data	116
7.5	Analysis with Call Stack Samples	116
7.5.1	Flame Graph	117
7.5.2	Call Graph	118
7.6	Profiling a Java Application	119
7.6.1	Launching a Java Application	119
7.6.2	Attaching a Java Process to Profile	120
7.6.3	Java Source View	120
7.6.4	Java Call Stack and Flame Graph	121
7.7	Cache Analysis	122
7.7.1	Supported Metrics	123
7.7.2	Cache Analysis Using GUI	123
7.7.3	Cache Analysis Using CLI	124
7.8	Custom Profile	126
7.8.1	Configuring and Starting Profile	126
7.8.2	Analyzing Profile Data	127
7.9	Advisory	128

7.9.1	Confidence Threshold	128
7.9.2	Issue Threshold	128
7.10	ASCII Dump of IBS Samples	129
7.11	Limitations	129
Chapter 8	Performance Analysis (Linux)	130
8.1	OpenMP Analysis	130
8.1.1	Profiling OpenMP Application using GUI	131
8.1.2	Profiling OpenMP Application Using CLI	132
8.1.3	Environment Variables	134
8.1.4	Limitations	134
8.2	MPI Profiling	134
8.2.1	Collecting Data Using CLI	135
8.2.2	Analyzing the Data with CLI	137
8.2.3	Analyze the Data with GUI	137
8.2.4	Limitations	137
8.3	Profiling Support on Linux for perf_event_paranoid Values	138
8.4	Profiling Linux System Modules	138
8.5	Profiling Linux Kernel	138
8.5.1	Enabling Kernel Symbol Resolution	139
8.5.2	Downloading and Installing Kernel Debug Symbol Packages	139
8.5.3	Build Linux kernel with Debug Symbols	140
8.5.4	Analyzing Hotspots in Kernel Functions	141
8.5.5	Linux Kernel Callstack Sampling	141
8.5.6	Constraints	141
8.6	Holistic Analysis	141
8.6.1	Holistic Analysis Using CLI	143
8.7	Thread State Analysis	146
8.7.1	Thread State Analysis Using CLI	146
8.8	Kernel Block I/O Analysis	149
8.8.1	Kernel Block I/O Analysis Using CLI	149
8.9	GPU Offloading Analysis (GPU Tracing)	151

8.9.1	GPU Offload Analysis Using CLI	152
8.10	GPU Profiling	155
8.10.1	GPU Profiling Using CLI	156
8.11	Other OS Tracing Events	158
8.11.1	Tracing Page Faults and Memory Allocations Using CLI	158
8.11.2	Tracing Function Call Count using CLI	159
8.12	MPI Trace Analysis	160
8.12.1	MPI Light-weight Tracing Using CLI	161
8.12.2	MPI Full Tracing Using CLI	162
8.12.3	MPI FULL Tracing Using GUI	168
Chapter 9	Power Profile	170
9.1	Overview	170
9.2	Metrics	170
9.3	Using Profile through GUI	172
9.3.1	Configuring a Profile	172
9.3.2	Analyzing a Profile	173
9.4	Using CLI to Profile	174
9.4.1	Examples	175
9.5	AMDPowerProfileAPI Library	176
9.5.1	Using the APIs	176
9.6	Limitations	177
Chapter 10	Energy Analysis	178
10.1	Overview	178
10.2	Using CLI to Profile	178
10.3	Limitations	179
Chapter 11	Remote Profiling	180
11.1	Overview	180
11.2	Setting up Authorization	180
11.3	Launching AMDProfilerService	181
11.4	Connecting to Remote Target	182
11.5	Limitations	183

Chapter 12	AMD uProf Virtualization Support	185
12.1	OverView	185
12.2	CPU Profiling	186
12.2.1	Profiling of Guest VM from Guest VM	186
12.2.2	Profiling of Guest VM from Host System (KVM Hypervisor)	187
12.2.3	Preparing Host system to Profile Guest Kernel Modules	187
12.2.4	AMD uProf CLI with Profiling Options	187
12.2.5	Examples	188
12.3	AMDuProfPcm	189
12.4	AMDuProfSys	189
Chapter 13	Profile Control APIs	190
13.1	AMDProfileControl APIs	190
13.1.1	CPU Profile Control APIs	190
13.1.2	Using the APIs	191
13.1.3	Compiling Instrumented Target Application	192
13.1.4	Profiling Instrumented Target Application	192
13.1.5	Limitations	192
Chapter 14	Reference	193
14.1	Preparing an Application for Profiling	193
14.1.1	Generating Debug Information on Windows	193
14.1.2	Generating Debug Information on Linux	194
14.2	CPU Profiling	194
14.2.1	Hardware Sources	195
14.2.2	Profiling Concepts	196
14.2.3	Profile Types	197
14.2.4	Predefined Core PMC Events	198
14.2.5	IBS Derived Events	206
14.3	Useful URLs	212

List of Tables

Table 1.	Conventions.	17
Table 2.	Abbreviations	17
Table 3.	Terminology	18
Table 4.	User Interface	20
Table 5.	Sampled Data	27
Table 6.	Predefined Sampling Configurations	28
Table 7.	Assess Performance Configurations	30
Table 8.	Investigate Data Access Configurations	30
Table 9.	Investigate Branch Configurations	30
Table 10.	Assess Performance (Extended) Configurations.	31
Table 11.	Investigate Instruction Access Configurations	31
Table 12.	Investigate CPI Configurations.	31
Table 13.	Instruction Based Sampling Configurations	32
Table 14.	AMDuProfPcm Options	35
Table 15.	Performance Metrics for AMD EPYC™ “Zen 2”	38
Table 16.	Performance Metrics for AMD EPYC™ “Zen 3”	41
Table 17.	Performance Metrics for AMD EPYC™ “Zen 4”	44
Table 18.	Level-1 Metrics.	54
Table 19.	Level-2 Metrics.	54
Table 20.	AMDuProfSys Generic Options	58
Table 21.	AMDuProfSys Collect Command Options.	58
Table 22.	AMDuProfSys Report Command Options	59
Table 23.	Supported Commands.	85
Table 24.	AMDuProfCLI Collect Command Options	89
Table 25.	AMDuProfCLI Collect Command – Windows Specific Options.	92
Table 26.	AMDuProfCLI Collect Command – Linux Specific Options.	93
Table 27.	AMDuProfCLI Report Command Options.	99
Table 28.	AMDuProfCLI Report Command - Windows Specific Options	101
Table 29.	AMDuProfCLI Report Command - Linux Specific Options	102

Table 30.	AMDuProfCLI Translate Command Options	104
Table 31.	Translate Command - Windows Specific Options	105
Table 32.	Translate Command - Linux Specific Options	105
Table 33.	AMDuProfCLI Timechart Command Options	107
Table 34.	AMDuProfCLI Info Command Options	109
Table 35.	AMDuProfCLI Info Command - Linux Specific Options	109
Table 36.	IBS OP Derived Metrics	123
Table 37.	Sort-by Metric	125
Table 38.	Support Matrix	130
Table 39.	MPI Profiling Support Matrix	135
Table 40.	Profiling perf_event_paranoid Values on Linux	138
Table 41.	Supported Events for Holistic Analysis	142
Table 42.	I/O Operations	149
Table 43.	Supported Interfaces for GPU Tracing	151
Table 44.	Supported Events for GPU Profiling	155
Table 45.	Supported Metrics for GPU Profiling	156
Table 46.	Supported Events for OS Tracing	158
Table 47.	List of Supported MPI APIs for Light-weight Tracing	161
Table 48.	MPI APIs	164
Table 49.	Family 17h Model 00h – 0Fh (AMD Ryzen™, AMD Ryzen ThreadRipper™, and 1 st Gen AMD EPYC™) 170	
Table 50.	Family 17h Model 10h – 1Fh (AMD Ryzen™ and AMD Ryzen™ PRO APU)	171
Table 51.	Family 17h Model 70h – 7Fh (3 rd Gen AMD Ryzen™)	171
Table 52.	Family 17h Model 30h – 3Fh (EPYC 7002)	171
Table 53.	Family 19h Model 0h – 2Fh (EPYC 7003 and EPYC 9000)	172
Table 54.	AMDProfilerService Options	181
Table 55.	AMD uProf Virtualization Support	185
Table 56.	AMD uProf CLI Collect Command Options	187
Table 57.	Predefined Core PMC Events	198
Table 58.	Core CPU Metrics	206
Table 59.	IBS Fetch Events	207

Table 60.	IBS Op Events.....	208
-----------	--------------------	-----

List of Figures

Figure 1.	Sample Roofline Chart	53
Figure 2.	Sample Report.	55
Figure 3.	AMD uProf GUI	62
Figure 4.	AMD uProf Welcome Screen	63
Figure 5.	Start Profiling - Select Profile Target	65
Figure 6.	Start Profiling - Select Profile Type	66
Figure 7.	Start Profiling - Advanced Options 1	67
Figure 8.	Start Profiling - Advanced Options 2	68
Figure 9.	Profile Data Collection	69
Figure 10.	Summary - Hot Spots Screen	70
Figure 11.	Summary - Thread Concurrency Graph	71
Figure 12.	ANALYZE - Function HotSpots	72
Figure 13.	Analyze - Metrics	73
Figure 14.	SOURCES - Source and Assembly	75
Figure 15.	ANALYZE - Flame Graph	77
Figure 16.	ANALYZE - Call Graph.	78
Figure 17.	IMIX View	79
Figure 18.	Import Session – Importing Profile Database.	80
Figure 19.	PROFILE - Recent Session(s)	81
Figure 20.	PROFILE - Saved Configurations	82
Figure 21.	SETTINGS - Preferences	83
Figure 22.	SETTINGS - Symbols	83
Figure 23.	SETTINGS - Source Data.	84
Figure 24.	Profile Data Cache	84
Figure 25.	Collect and Report Commands	86
Figure 26.	List of Supported Predefined Configurations	87
Figure 27.	Output of timechart --list Command.	88
Figure 28.	Execution of timechart	89
Figure 29.	Time-based Profile – Configure	113

Figure 30.	Event-based Profile – Configure.	114
Figure 31.	Start Profiling - Advanced Options	117
Figure 32.	ANALYZE - Flame Graph	118
Figure 33.	ANALYZE - Call Graph.	119
Figure 34.	Java Method - Source View	121
Figure 35.	Java Application - Flame Graph	122
Figure 36.	Cache Analysis	124
Figure 37.	Cache Analysis - Summary Sections	125
Figure 38.	Cache Analysis - Detailed Report.	125
Figure 39.	Start Profiling - Custom Profile	127
Figure 40.	CPI Metric - Threshold-based Performance Issue	128
Figure 41.	Enable OpenMP Tracing	131
Figure 42.	HPC - Overview	131
Figure 43.	HPC - Parallel Regions	132
Figure 44.	An OpenMP Report	133
Figure 45.	System Analysis - OS Tracing Report	144
Figure 46.	System Analysis - TIMECHART	145
Figure 47.	Thread State Analysis - OS Tracing Report	147
Figure 48.	Thread State Analysis - TIMECHART	148
Figure 49.	Disk I/O Summary Tables	150
Figure 50.	ANALYZE - Block I/O Stats	151
Figure 51.	GPU Tracing Report	153
Figure 52.	TIMECHART - GPU Offload Analysis	154
Figure 53.	GPU Profile Report.	157
Figure 54.	Monitored Events	159
Figure 55.	Function Count Summary	159
Figure 56.	LWT Report	162
Figure 57.	MPI Communicator Summary Table	166
Figure 58.	MPI Rank Summary Table	166
Figure 59.	MPI Point-to-Point API Summary Table	166
Figure 60.	MPI Communication Matrix.	167

Figure 61.	MPI Collective API Summary Table	167
Figure 62.	Import Profile Session	168
Figure 63.	MPI Communication Matrix.....	168
Figure 64.	TIMECHART - MPI Rank Analysis	169
Figure 65.	Live System-wide Power Profile	173
Figure 66.	Timechart Page	174
Figure 67.	--list Command Output	175
Figure 68.	Timechart Run	175
Figure 69.	Client ID	180
Figure 70.	Remote Profiling Connection Establishment	181
Figure 71.	Selecting IP	182
Figure 72.	Connect to Remote Machine.....	182
Figure 73.	Remote Target Data	183
Figure 74.	Disconnect Button.....	183
Figure 75.	AMDTClassicMatMul Property Page	194

Revision History

Date	Revision	Description
November 2022	4.0	Included AMD uProf 4.0 features
July 2022	3.6	Added the following: <ul style="list-style-type: none">• Chapters 11 and 12• Sections 1.2.4, 1.2.5, 3.4, 4.2.1, 4.3, 5.4.8, 6.6, 8.10.2, and 13.1.5 Deleted Supported Counter categories for older APU families in chapter 9 Performed general edits and included release related updates
January 2022	3.5	Included AMD uProf 3.5 features
April 2021	Initial	Documented AMD uProf 3.4 features

About this Document

This document describes how to use AMD uProf to perform CPU, GPU, and power analysis of applications running on Windows®, Linux®, and FreeBSD® operating systems on AMD processors.

The latest version of this document is available in the AMD uProf web site (<https://developer.amd.com/amd-uprof/>).

Intended Audience

This document is intended for the 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

The following conventions have been used in this document:

Table 1. Conventions

Convention	Description
GUI element	A Graphical User Interface element such as 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
<i>File name</i>	Name of a file or path or source code snippet
Command	Command name or command phrase
<i>Hyperlink</i>	Links to external web sites

Abbreviations

The following abbreviations have been used in this document:

Table 2. Abbreviations

Abbreviation	Description
ASLR	Address Space Layout Randomization
CCD	Core Complex Die that can contain one or more CCX(s) and GMI2 Fabric port(s) connecting to IOD
CLI	Command Line Interface

Table 2. Abbreviations

Abbreviation	Description
CPI	Cycles Per Instruction
CSV	Comma Separated Values format
DC	Data Cache
DIMM	Dual In-line Memory Module
DRAM	Dynamic Random Access Memory
DTLB	Data Translation Lookaside Buffer
EBP	Event Based Profiling, uses Core PMC events
GUI	Graphical User Interface
IBS	Instruction Based Sampling
IC	Instruction Cache
IOD	IO Die
IPC	Instructions Per Cycle
ITLB	Instruction Translation Lookaside Buffer
MSR	Model Specific Register
NB	Northbridge
OS	Operating System
PMC	Performance Monitoring Counter
PTI	Per Thousand Instructions
RAPL	Running Average Power Limit
SMU	System Management Unit
TBP	Timer Based Profiling
UMC	Unified Memory Controllers Up to 8 UMCs, each supporting one DRAM channel per socket; each channel can have up to 2 DIMMs

Terminology

The following terms have been used in this document:

Table 3. Terminology

Term	Description
AMD uProf	The product name uProf.
AMDuProfGUI	The name of the graphical user interface tool.
AMDuProfCLI	The name of the command line interface tool.

Table 3. Terminology

Term	Description
AMDuProfPcm	The name of the command line interface tool for System Analysis.
AMDuProfSys	The name of the python based command line interface tool for System Analysis.
Client	Instance of AMD uProf or AMDuProfCLI running on a host system.
Core	The logical core number, a core can contain one or two CPU(s) depending on the SMT configuration.
Core Complex (CCX)	Consists of one or many cores and a cache system.
CPU	Logical CPU numbers as considered by the operating system.
Host system	System in which the AMD uProf client process runs.
L1D, L1I Cache	CPU exclusive data and instruction cache.
L2 Cache	Shared by all the CPUs within the core.
L3 Cache	Shared by all the CPUs within CCX.
Node	Logical NUMA node.
Performance Profiling (or) CPU Profiling	Identify and analyze the performance bottlenecks. Performance Profiling and CPU Profiling denotes the same.
Socket	The logical socket number, a socket can contain multiple nodes.
System Analysis	Refers to AMDuProfPcm or AMDuProfSys tools.
Target system	System in which the profile data is collected.

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 understand and improve the runtime performance of their application.

AMD uProf offers the following functionalities:

- **Performance Analysis (CPU Profile)**
To identify runtime performance bottlenecks of the application.
- **System Analysis**
To monitor system performance metrics, such as IPC and 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 the following user interfaces:

Table 4. User Interface

Executable	Description	Supported OS
AMDuProf	GUI to perform CPU and Power Profile	Windows and Linux
AMDuProfCLI	CLI to perform CPU and Power Profile	Windows, Linux, and FreeBSD
AMDuProfPcm	CLI to perform System Analysis	Windows, Linux, and FreeBSD
AMDPerf/ AMDuProfSys.py	Python script for System Analysis	Windows and 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 kernels, drivers, and system modules.
- Observe system level thermal and power characteristics.
- Observe system metrics, such as IPC and memory bandwidth.

1.2 Specification

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

1.2.1 Processors

- AMD “Zen”-based CPU and APU Processors
- AMD Instinct™ MI100 and MI200 accelerators (for GPU kernel profiling and tracing)
- Intel® Processors (Timer based profiling only)

1.2.2 Operating Systems

AMD uProf supports the 64-bit versions of the following operating systems:

- Microsoft
 - Windows 10 and 11
 - Windows Server 2019 and 2022
- Linux
 - Ubuntu 16.04 and later
 - RHEL 7.0 and later
 - CentOS 7.0 and later
 - openSUSE Leap 15.0
 - SLES 12 and 15
- FreeBSD 12.2 and later

For OS support on AMD EPYC™ 7003 Series processors, refer to AMD website (<https://www.amd.com/en/processors/epyc-minimum-operating-system>).

1.2.3 Compilers and Application Environment

AMD uProf supports the following application environments:

- Languages
 - Native languages: C, C++, Fortran, and Assembly
 - Non-native languages: Java and C#
- Programs compiled with
 - Microsoft compilers, GNU compilers, and LLVM
 - AMD Optimizing CPU Compilers (AOCC)
 - Intel Compilers (ICC)

- Parallelism
 - OpenMP
 - MPI
- Debug info formats: PDB, COFF, DWARF, and STABS
- Applications compiled with and without optimization or debug information
- Single-process, multi-process, single-thread, and multi-threaded applications
- Dynamically linked/loaded libraries
- POSIX development environment on Windows
 - Cygwin
 - MinGW

1.2.4 Virtualization Support

AMD uProf can be used on virtualized environments. There could be limitations related to access to hardware performance counters. For more information, refer to “AMD uProf Virtualization Support” on page 185. The following virtualized environments are supported:

- VMware ESXi
- Linux KVM
- Citrix Xen
- Microsoft Hyper-V

1.2.5 Container Support (Beta Support)

AMD uProf can be used in Docker container environments. The following scenarios are supported:

- Run AMD uProf inside the Docker container to analyze the running target application. CAP_SYS_ADMIN permission is required for the Docker container.
- Run AMD uProf outside the container to analyze the target application running in a container:
 - a. Attach to the containerized process using the `--pid` option during collection.
 - b. Collect the system-wide data and then filter by PID during report generation.

1.3 Installing AMD uProf

Download the latest version of the AMD uProf installer package for the supported operating systems from AMD Developer Central (<https://developer.amd.com/amd-uprof/>). You can install it using one of the following methods.

1.3.1 Windows

Run the 64-bit Windows installer binary *AMDuProf-x.y.z.exe*.

After the installation is complete, the executables, libraries, and the other required files are installed in the folder *C:\Program Files\AMD\AMDuProf*.

1.3.2 Linux

Installing Using a tar File

Extract the tar.bz2 binary file and install AMD uProf using the following command:

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

Note: *The Power Profiler Linux Driver must be installed manually.*

Installing Using a RPM Package (RHEL)

Install the AMD uProf RPM package by 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
```

After the installation is complete, the executables, libraries, and the other required files will be installed in the directory */opt/AMDuProf_X.Y-ZZZ/*.

Installing Using a Debian Package (Ubuntu)

Install the AMD uProf Debian package by using the **dpkg** command:

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

After the installation is complete, the executables, libraries, and the other required files will be installed in the directory */opt/AMDuProf_X.Y-ZZZ/*.

Installing Power Profiling Driver on Linux

While installing AMD uProf using RPM and Debian installer packages, the Power Profiler Linux Driver build is generated and installed automatically. However, if you downloaded the AMD uProf tar.bz2 archive, you must install the Power Profiler Linux Driver manually.

The GCC and MAKE software packages are prerequisites for installing Power Profiler Driver. If you do not have these packages, you can install them using the following commands:

On RHEL and CentOS distros:

```
$ sudo yum install gcc make
```

On Debian/Ubuntu distros:

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

Execute the following commands:

```
$ 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 in the directory `/usr/src/AMDPowerProfiler-<version>`. All the source files required for module compilation are in this directory and under MIT license.

To uninstall the driver run the following commands:

```
$ 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 Profiler Driver module whenever there is a change in the existing kernel. This saves you from manually upgrading the power profiling driver module. The DKMS package must be installed on target machines before running the installation steps mentioned in the above section.

`AMDPowerProfilerDriver.sh` installer script will automatically handle the DKMS related configuration if the DKMS package is installed on 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 you upgrade the kernel version frequently, it is recommended to use DKMS for the installation.

Installing ROCm

Complete the steps in the ROCm installation guide (https://docs.amd.com/bundle/ROCm-Installation-Guide-v5.2.1/page/Introduction_to_ROCm_Installation_Guide_for_Linux.html) to install AMD ROCm™ v5.2.1 on the host system.

After ROCm 5.2.1 installation, make sure symbolic link of `/opt/rocm/` points to `/opt/rocm-5.2.1/`.

```
$ ln -s /opt/rocm-5.2.1/ /opt/rocm/
```

AMD ROCm v5.2.1 installation is required for GPU tracing and profiling.

Installing BCC and eBPF

Complete the steps on the BCC website (<https://github.com/iovisor/bcc/blob/master/INSTALL.md>) to install it.

After installing BCC, run the following command to validate the BCC installation:

```
$ cd AMDuProf_Linux_x64_x.y.z/bin
$ sudo ./AMDuProfVerifyBpfInstallation.sh
```


If you install AMD uProf using RPM/DEB installer, the script is run by the installer and the info about BCC installation and eBPF (Extended Berkeley Packet Filter) support on the host is provided.

1.3.3 FreeBSD

Extracting the *tar.bz2* binary file and install AMD uProf:

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

1.4 Sample Programs

A few sample programs are installed along with the product for you to 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

For support options, the latest documentation, and downloads refer AMD Developer Central (<https://developer.amd.com/amd-uprof/>).

Chapter 2 Workflow and Key Concepts

2.1 Workflow

The AMD uProf workflow has the following phases:

1. Collect — Run the application program and collect the profile data.
2. Translate — Process the profile data to aggregate, correlate, and organize into database.
3. Analyze — View and analyze the performance data to identify the bottlenecks.

2.1.1 Collect Phase

Important concepts of the collect phase are explained in this section.

Profile Target

The profile target is one 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 a running application (native applications only).

Profile Type

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

- CPU Profile
- GPU Profile
- GPU Trace
- System-wide Power Profile
- Energy Analysis (Windows only)

The data collection is defined by **Sampling Configuration**:

- **Sampling Configuration** identifies the set of Sampling Events, 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.

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).

The following table shows the type of profile data collected and sampling events for a profile type:

Table 5. Sampled Data

Profile Type	Type of Profile Data Collected	Sampling Events
CPU Profiling	<ul style="list-style-type: none"> • Process ID • Thread ID • IP • Callstack • ETL tracing (Windows only) • OpenMP Trace — OMPT (Linux) • MPI Trace — PMPI (Linux) • OS Trace — Linux BPF 	<ul style="list-style-type: none"> • OS Timer • Core PMC events • IBS
GPU Profiling	Perfmon Metrics	Not applicable
GPU Tracing	Runtime Trace — HIP and HSA	Not applicable

For CPU Profiling, there are numerous micro-architecture specific events available to monitor. The tool groups the related and interesting events to monitor called **Predefined Sampling Configuration**. For example, **Assess Performance** is one such configuration used to get the overall assessment of the performance and to find potential issues for investigation. For more information, refer “Predefined View Configuration” on page 29.

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

Profile Configuration

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

The GUI saves these profile configuration details with a default name (for example, AMDuProf-TBP-Classic), you can define them too. As the performance analysis is iterative, this is persistent (can be deleted) and hence, you can also reuse the same configuration for the 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 and translated data (in a database) in the folder named as <profile config name>-<timestamp>.

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

2.1.2 Translate Phase

The collected raw profile data is processed to aggregate and attribute to the respective processes, threads, load modules, functions, and instructions. The 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 the GUI after the profiling is stopped. In the CLI, the report command implicitly processes (translates) the raw profile data and generates the report in CSV format. Also, the CLI provides translate command to perform only the translation and the translated data files can be imported to GUI for visualization.

2.1.3 Analyze Phase

View Configuration

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

The tool can be used to filter/view only specific configurations, which is called **Predefined View**. For example, **IPC assessment** view lists metrics such as CPU Clocks, Retired Instructions, IPC, and CPI. For more information, refer “Predefined Sampling Configuration” on page 28.

2.2 Predefined Sampling Configuration

The **Predefined Sampling Configuration** provides a convenient way to select a useful set of sampling events for profile analysis. The following table lists all such configurations:

Table 6. Predefined Sampling Configurations

Profile Type	Predefined Configuration Name	Abbreviation	Description
Time-based profile (TBP)	Time-based profile	tbp	To identify where the programs are consuming time.

Table 6. Predefined Sampling Configurations

Profile Type	Predefined Configuration Name	Abbreviation	Description
Event-based profile (EBP)	Assess performance	assess	Provides an overall assessment of the performance.
	Assess performance (Extended)	assess_ext	Provides an overall assessment of the 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.
	Investigate CPI	cpi	To analyze the CPI and IPC metrics of the running application or the entire system.
IBS	Instruction based sampling	ibs	To collect the sample data using IBS Fetch and IBS OP. Precise sample attribution to instructions.
	Cache Analysis	memory	To identify the false cache-line sharing issues. The profile data will be collected using IBS OP.
Energy	Energy analysis	energy	To identify where the programs are consuming energy.

Notes:

1. The AMDuProf GUI uses the **name** of the predefined configuration in the above table.
2. The abbreviation (in Table 6 on page 28) is used with AMDuProfCLI **collect** command's **--config** option.
3. The supported predefined configurations and the sampling events used in them is based on the processor family and model.

2.3 Predefined View Configuration

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

Following is the list of predefined view configurations for **Assess Performance**:

Table 7. Assess Performance Configurations

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

The following table lists the predefined view configurations for **Investigate Data Access**:

Table 8. Investigate Data Access Configurations

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

The following table lists the predefined view configurations for **Investigate Branch**:

Table 9. Investigate Branch Configurations

View configuration	Abbreviation	Description
Investigate Branching	Branch	You can use this view to find code with a high branch density and poorly predicted branches.
IPC assessment	ipc_assess	Find hotspots with low instruction level parallelism, provides performance indicators – IPC and CPI.
Branch assessment	br_assess	You can use this view to find code with a high branch density and poorly predicted branches.

Table 9. Investigate Branch Configurations

View configuration	Abbreviation	Description
Taken branch report	taken_focus	You can use this view to find the code with a high number of taken branches.
Near return report	return_focus	You can use this view to find code with poorly predicted near returns.

The following table lists the predefined view configurations for **Assess Performance (Extended)**:

Table 10. Assess Performance (Extended) Configurations

View configuration	Abbreviation	Description
Assess Performance (Extended)	triage_assess_ext	This view gives an overall picture of performance. You can use it to find possible issues for deeper investigation.
IPC assessment	ipc_assess	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	Provides information about data cache (DC) access including DC miss rate and DC miss ratio.
Misaligned access assessment	misalign_assess	Identify regions of code that access misaligned data.

The following table lists the predefined view configurations for **Investigate Instruction Access**:

Table 11. Investigate Instruction Access Configurations

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

The following table lists the predefined view configurations for **Investigate CPI**:

Table 12. Investigate CPI Configurations

View configuration	Abbreviation	Description
IPC assessment	ipc_assess	Find hotspots with low instruction level parallelism. Provides performance indicators – IPC and CPI.

The following table lists the predefined view configurations for **Instruction Based Sampling**:

Table 13. Instruction Based Sampling Configurations

View configuration	Abbreviation	Description
IBS fetch overall	ibs_fetch_overall	You can use this view to display an overall summary of the IBS fetch sample data.
IBS fetch instruction cache	ibs_fetch_ic	You can use this view to display a summary of IBS attempted fetch Instruction Cache (IC) miss data.
IBS fetch instruction TLB	ibs_fetch_itlb	You can use this view to display a summary of IBS attempted fetch ITLB misses.
IBS fetch page translations	ibs_fetch_page	You can use this view to display a summary of the IBS L1 ITLB page translations for attempted fetches.
IBS All ops	ibs_op_overall	You can use this view to display a summary of all IBS Op samples.
IBS MEM all load/store	ibs_op_ls	You can use this view to display a summary of IBS Op load/store data.
IBS MEM data cache	ibs_op_ls_dc	You can use this view to display a summary of DC behavior derived from IBS Op load/store samples.
IBS MEM data TLB	ibs_op_ls_dtlb	You can use this view to display a summary of DTLB behavior derived from IBS Op load/store data.
IBS MEM locked ops and access by type	ibs_op_ls_memacc	You can use this view to display the uncacheable (UC) memory access, write combining (WC) memory access, and locked load/store operations.
IBS MEM translations by page size	ibs_op_ls_page	You can use this view to display a summary of DTLB address translations broken out by page size.
IBS MEM forwarding and bank conflicts	ibs_op_ls_expert	You can use this view to display the memory access bank conflicts, data forwarding, and Missed Address Buffer (MAB) hits.
IBS BR branch	ibs_op_branch	You can use this view to display the IBS retired branch op measurements including mispredicted and taken branches.
IBS BR return	ibs_op_return	You can use this view to display the IBS return op measurements including the return misprediction ratio.
IBS NB local/remote access	ibs_op_nb_access	You can use this view to display the number and latency of local and remote accesses.
IBS NB cache state	ibs_op_nb_cache	You can use this view to display the cache owned (O) and modified (M) state for NB cache service requests.
IBS NB request breakdown	ibs_op_nb_service	You can use this view to display the breakdown of NB access requests.

Notes:

1. The AMDuProf GUI uses the 'View configuration' **name** of the predefined configuration mentioned in the above table.
2. The abbreviation is used in the CLI generated report file.
3. The supported predefined configurations and the sampling events used in them is based on the processor family and model.

Chapter 3 Getting started with AMDuProfPcm – System Analysis

3.1 Overview

The System Analysis utility AMDuProfPcm helps to monitor basic performance monitoring metrics for AMD EPYC™ 7001, AMD EPYC™ 7002, and AMD EPYC™ 7003 of family 17h and 19h processors. This utility periodically collects the CPU Core, L3, and DF performance event count values and reports various metrics. It is supported on Windows, Linux, and FreeBSD.

3.1.1 Prerequisite(s)

3.1.1.1 Linux

- AMDuProfPcm uses msr driver and either requires root privileges or read write permissions for `/dev/cpu/*/msr` devices.
- NMI watchdog must be disabled (`echo 0 > /proc/sys/kernel/nmi_watchdog`).
- Set `/proc/sys/kernel/perf_event_paranoid` to -1.
- Use the following command to load the msr driver:

```
$ modprobe msr
```

3.1.1.2 FreeBSD

AMDuProfPcm uses cpuctl module and requires either root privileges or read write permissions for `/dev/cpuctl*` devices.

Synopsis:

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

<PROGRAM> — Denotes the 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
```

3.2 Options

The following table lists all the options:

Table 14. AMDuProfPcm Options

Option	Description
-h	Displays this help information on the console/terminal.
-m <metric,...>	<p>Metrics to report, the default metric group is 'ipc'.</p> <p>The supported metric groups and the corresponding metrics are Platform, OS, and Hypervisor specific.</p> <p>Run AMDuProfpcm -h to get the list of supported metrics.</p> <p>The following metric groups are supported:</p> <ul style="list-style-type: none"> • ipc – reports metrics such as CEF, Utilization, CPI, and 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 such as DC refills by source (supported only on AMD “Zen3” and AMD “Zen4” processors) • memory – approximate memory read and write bandwidths in GB/s for all the channels • pcie – PCIe bandwidth in GB/s (supported only on AMD “Zen2” processors) • xgmi – approximate xGMI outbound databytes in GB/s for all the remote links • dma – DMA bandwidth in GB/s (supported only on AMD “Zen4” processors)

Table 14. AMDuProfPcm Options

Option	Description
-c <core ccx ccd package=<n>	<p>Collect from the specified core ccx ccd package. The default is 'core=0'.</p> <p>If 'ccx' is specified:</p> <ul style="list-style-type: none"> • The core events will be collected from all the cores of this ccx. • The l3 and df events will be collected from the first core of this ccx. <p>If 'ccd' is specified:</p> <ul style="list-style-type: none"> • The core events will be collected from all the cores of this die. • The l3 events will be collected from the first core of all the ccx's of this die. • The df events will be collected from the first core of this die. <p>If 'package' is specified:</p> <ul style="list-style-type: none"> • The core events will be collected from all the cores of this package. • The l3 events will be collected from the first core of all the ccx's of this package. • The df events will be collected from the first core of all the die of this package.
-a	<p>Collect from all the cores.</p> <p><i>Note: Options -c and -a cannot be used together.</i></p>
-C	<p>Prints the cumulative data at the end of the profile duration. Otherwise, all the samples will be reported as timeseries data.</p>
-A <system,package,ccd,ccx,core>	<p>Prints aggregated metrics at various component level. The following granularities are supported:</p> <ul style="list-style-type: none"> • 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. <p><i>Notes:</i></p> <ol style="list-style-type: none"> 1. Option -a should be used along with this option to collect samples from all the cores. 2. Comma separated list of components can be specified.

Table 14. AMDuProfPcm Options

Option	Description
-i <config file>	User defined XML config file that specifies Core L3 DF counters to monitor. Refer sample files in <install-dir>/bin/Data/Config/ dir for the format. <i>Notes:</i> 1. Options -i and -m cannot be used together. 2. 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>	The interval in which pmc count values will be read, the minimum is 16 ms.
-o <output file>	The output file name, it is in CSV format.
-D <dump file>	The output file that contains the event count dump for all the monitored events. It is in CSV format.
-p <n>	Sets precision of the metrics reported, the default value is 2.
-q	Hide CPU topology section in the output report.
-r	Force resets the MSRs.
-k	Prefixes 'pkg' in package level counters.
-s	Displays time stamp in the time series report.
-l	Lists the supported raw PMC events.
-z <pmc-event>	Prints the name, description, and available unit masks for the event.
-x <core-id,...>	Core affinity for launched application, comma separated list of core IDs. <i>Note: This is supported only on Linux.</i>
-w <dir>	Specifies the working directory. The default will be the path of the launched application.
-v	Print version.

Following are the performance metrics for AMD EPYC™ “Zen 2” core architecture processors:

Table 15. Performance Metrics for AMD EPYC™ “Zen 2”

Metric Group	Metric	Description
ipc	Utilization (%)	Percentage of time the core was running, that is non-idle time.
	Eff Freq	Core Effective Frequency (CEF) 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	Instructions 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. A lower CPI value is better.
	Branch Mis-prediction Ratio	The ratio between mis-predicted 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).
ll	IC(32B) Fetch Miss Ratio	Instruction cache fetch miss ratio.
	DC Access	All data cache (DC) accesses. This metric is in PTI.

Table 15. Performance Metrics for AMD EPYC™ “Zen 2”

Metric Group	Metric	Description
l2	L2 Access	All the L2 cache accesses. This metric is in PTI.
	L2 Access from IC Miss	The L2 cache accesses from IC miss. This metric is in PTI.
	L2 Access from DC Miss	The L2 cache accesses from DC miss. This metric is in PTI.
	L2 Access from HWPF	The L2 cache accesses from L2 hardware pre-fetching. This metric is in PTI.
	L2 Miss	All the L2 cache misses. This metric is in PTI.
	L2 Miss from IC Miss	The L2 cache misses from IC miss. This metric is in PTI.
	L2 Miss from DC Miss	The L2 cache misses from DC miss. This metric is in PTI.
	L2 Miss from HWPF	The L2 cache misses from L2 hardware pre-fetching. This metric is in PTI.
	L2 Hit	All the L2 cache hits. This metric is in PTI.
	L2 Hit from IC Miss	The L2 cache hits from IC miss. This metric is in PTI.
	L2 Hit from DC Miss	The L2 cache hits from DC miss. This metric is in PTI.
	L2 Hit from HWPF	The L2 cache hits from L2 hardware pre-fetching. This metric is in PTI.
tlb	L1 ITLB Miss	The instruction fetches the 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 PTI.
	L2 ITLB Miss	The number of ITLB reloads from page table walker due to L1-ITLB and L2-ITLB misses. This metric is in 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 PTI.
	L2 DTLB Miss	The number of L2 Data Translation Lookaside Buffer (DTLB) missed from load store micro-ops. This metric is in PTI.

Table 15. Performance Metrics for AMD EPYC™ “Zen 2”

Metric Group	Metric	Description
l3	L3 Access	The L3 cache accesses. This metric is in PTI.
	L3 Miss	The L3 cache miss. This metric is in PTI.
	L3 Miss (%)	The L3 cache miss percentage. This metric is in 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 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 are the performance metrics for AMD EPYC™ “Zen 3” core architecture processors:

Table 16. Performance Metrics for AMD EPYC™ “Zen 3”

Metric Group	Metric	Description
ipc	Utilization (%)	Percentage of time the core was running, that is non-idle time.
	Eff Freq	Core Effective Frequency (CEF) 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	Instructions 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. A lower CPI value is better.
	Branch Mis-prediction Ratio	The ratio between mis-predicted 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).
ll	IC (32B) Fetch Miss Ratio	Instruction cache fetch miss ratio.
	Op Cache (64B) Fetch Miss Ratio	Operation cache fetch miss ratio.
	IC Access	All instruction cache accesses. This metric is in PTI.
	IC Miss	The instruction cache miss. This metric is in PTI.
	DC Access	All the DC accesses. This metric is in PTI.

Table 16. Performance Metrics for AMD EPYC™ “Zen 3”

Metric Group	Metric	Description
l2	L2 Access	All the L2 cache accesses. This metric is in PTI.
	L2 Access from IC Miss	The L2 cache accesses from IC miss. This metric is in PTI.
	L2 Access from DC Miss	The L2 cache accesses from DC miss. This metric is in PTI.
	L2 Access from HWPF	The L2 cache accesses from L2 hardware pre-fetching. This metric is in PTI.
	L2 Miss	All the L2 cache misses. This metric is in PTI.
	L2 Miss from IC Miss	The L2 cache misses from IC miss. This metric is in PTI.
	L2 Miss from DC Miss	The L2 cache misses from DC miss. This metric is in PTI.
	L2 Miss from HWPF	The L2 cache misses from L2 hardware pre-fetching. This metric is in PTI.
	L2 Hit	All the L2 cache hits. This metric is in PTI.
	L2 Hit from IC Miss	The L2 cache hits from IC miss. This metric is in PTI.
	L2 Hit from DC Miss	The L2 cache hits from DC miss. This metric is in PTI.
	L2 Hit from HWPF	The L2 cache hits from L2 hardware pre-fetching. This metric is in PTI.
tlb	L1 ITLB Miss	The instruction fetches the 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 PTI.
	L2 ITLB Miss	The number of ITLB reloads from page table walker due to L1-ITLB and L2-ITLB misses. This metric is in 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 PTI.
	L2 DTLB Miss	The number of L2 Data Translation Lookaside Buffer (DTLB) missed from load store micro-ops. This metric is in PTI.
	All TLBs Flushed	All the TLBs flushed. This metric is in PTI.

Table 16. Performance Metrics for AMD EPYC™ “Zen 3”

Metric Group	Metric	Description
dc	DC Fills from Same CCX	The number of 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 PTI.
	DC Fills from different CCX in same node	The number of DC fills from cache of different CCX in the same package (node). This metric is in PTI.
	DC Fills from Local Memory	The number of DC fills from DRAM or IO connected in the same package (node). This metric is in PTI.
	DC Fills from Remote CCX Cache	The number of DC fills from cache of CCX in the different package (node). This metric is in PTI.
	DC Fills from Remote Memory	The number of DC fills from DRAM or IO connected in the different package (node). This metric is in PTI.
	All DC Fills	The total number of DC fills from all the data sources. This metric is in PTI.
l3	L3 Access	The L3 cache accesses. This metric is in PTI.
	L3 Miss	The L3 cache miss. This metric is in PTI.
	L3 Miss (%)	The L3 cache miss percentage. This metric is in PTI.
	Ave L3 Miss Latency	The 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 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.

Following are the performance metrics for AMD EPYC™ “Zen 4” core architecture processors:

Table 17. Performance Metrics for AMD EPYC™ “Zen 4”

Metric Group	Metric	Description
ipc	Utilization (%)	Percentage of time the core was running, that is non-idle time.
	Eff Freq	Core Effective Frequency (CEF) 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	Instructions 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. A lower CPI value is better.
	Branch Mis-prediction Ratio	The ratio between mis-predicted branches and retired branch instructions.
fp	Retired SSE/AVX Flops (GFLOPs)	The number of retired SSE/AVX FLOPs.
	FP Dispatch Faults (PTI)	The floating point instruction dispatch fault. This metric is in per thousand instructions (PTI).
ll	IC (32B) Fetch Miss Ratio	Instruction cache fetch miss ratio.
	Op Cache Fetch Miss Ratio	Operation cache (64B) fetch miss ratio.
	IC Access (PTI)	Instruction cache access in PTI.
	IC Miss (PTI)	Instruction cache Miss in PTI.
	DC Access (PTI)	All the data cache (DC) accesses. This metric is in PTI.

Table 17. Performance Metrics for AMD EPYC™ “Zen 4”

Metric Group	Metric	Description
l2	L2 Access	All the L2 cache accesses. This metric is in PTI.
	L2 Access from IC Miss	The L2 cache accesses from IC miss. This metric is in PTI.
	L2 Access from DC Miss	The L2 cache accesses from DC miss. This metric is in PTI.
	L2 Access from HWPF	The L2 cache accesses from L2 hardware pre-fetching. This metric is in PTI.
	L2 Miss	All the L2 cache misses. This metric is in PTI.
	L2 Miss from IC Miss	The L2 cache misses from IC miss. This metric is in PTI.
	L2 Miss from DC Miss	The L2 cache misses from DC miss. This metric is in PTI.
	L2 Miss from HWPF	The L2 cache misses from L2 hardware pre-fetching. This metric is in PTI.
	L2 Hit	All the L2 cache hits. This metric is in PTI.
	L2 Hit from IC Miss	The L2 cache hits from IC miss. This metric is in PTI.
	L2 Hit from DC Miss	The L2 cache hits from DC miss. This metric is in PTI.
	L2 Hit from HWPF	The L2 cache hits from L2 hardware pre-fetching. This metric is in PTI.
tlb	L1 ITLB Miss	The instruction fetches the 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 PTI.
	L2 ITLB Miss	The number of ITLB reloads from page table walker due to L1-ITLB and L2-ITLB misses. This metric is in 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 PTI.
	L2 DTLB Miss	The number of L2 Data Translation Lookaside Buffer (DTLB) missed from load store micro-ops. This metric is in PTI.
	All TLBs Flushed	All the flushed TLBs.

Table 17. Performance Metrics for AMD EPYC™ “Zen 4”

Metric Group	Metric	Description
l3	L3 Access	The L3 cache accesses. This metric is in PTI.
	L3 Miss	The L3 cache miss. This metric is in PTI.
	L3 Miss (%)	The L3 cache miss percentage. This metric is in PTI.
	Ave L3 Miss Latency	Average L3 miss latency in core cycles.
Memory	Total Memory Bw (GB/s)	Total read and write memory bandwidth.
	Local DRAM Read Data Bytes (GB/s)	DRAM read and write data bytes for a local processor.
	Local DRAM Write Data Bytes (GB/s)	
	Remote DRAM Read Data Bytes (GB/s)	DRAM read and write data bytes for a remote processor.
	Remote DRAM Write Data Bytes (GB/s)	
	Mem Ch-A RdBw (GB/s) Mem Ch-A WrBw (GB/s) ...	Memory read and write bandwidth in GB/s for all the channels.
xgmi	Local Inbound Read Data Bytes (GB/s)	Local inbound data bytes to the CPU, for example, read data.
	Local Outbound Write Data Bytes (GB/s)	Local outbound data bytes from the CPU, for example, write data.
	Remote Inbound Read Data Bytes (GB/s)	Remote socket inbound data bytes to the CPU, for example, read data.
	Remote Outbound Write Data Bytes (GB/s)	Remote socket outbound data bytes from the CPU for example, write data.
	xGMI Outbound Data Bytes (GB/s)	Total outbound data bytes in Gigabytes per second.
dma	Total Upstream DMA Read Write Data Bytes (GB/s)	Total upstream DMA including read and write.
	Local Upstream DMA Read Data Bytes (GB/s)	Local upstream DMA read data bytes.
	Local Upstream DMA Write Data Bytes (GB/s)	Local upstream DMA write data bytes.
	Remote Upstream DMA Read Data Bytes (GB/s)	Remote socket upstream DMA read data bytes
	Remote Upstream DMA Write Data Bytes (GB/s)	Remote socket upstream DMA write data bytes.

Table 17. Performance Metrics for AMD EPYC™ “Zen 4”

Metric Group	Metric	Description
Top-down	Total_Dispatch_Slots	Up to 6 instructions can be dispatched in one cycle.
	SMT_Disp_contention	Fraction of unused dispatch slots as other thread was selected.
	Frontend_Bound	Fraction of dispatch slots that remained unused as the frontend did not supply enough instructions/operations.
	Bad_Speculation	Fraction of unused dispatch slots as other thread was selected.
	Backend_Bound	Fraction of dispatch slots that remained unused because of the backend stalls.
	Retiring	Fraction of dispatch slots used by the retired operations.
	IPC	Instructions per cycle.
	Frontend_Bound.Latency	Fraction of dispatch slots that remained unused because of a latency bottleneck in the frontend, such as Instruction Cache or ITLB misses.
	Frontend_Bound.BW	Fraction of dispatch slots that remained unused because of a bandwidth bottleneck in the frontend, such as decode bandwidth or Op Cache fetch bandwidth.
	Bad_Speculation.Mispredicts	Fraction of dispatched ops that were flushed due to branch mis-predicts.
	Bad_Speculation.Pipeline_Restarts	Fraction of dispatched ops that were flushed due to the pipeline restarts (resyncs).
	Backend_Bound.Memory	Fraction of dispatched slots that remained unused because of stalls due to the memory subsystem.
	Backend_Bound.CPU	Fraction of dispatched slots that remained unused because of stalls not related to the memory subsystem.
	Retiring.Fastpath	Fraction of dispatch slots used by the retired fastpath operations.
	Retiring.Microcode	Fraction of dispatch slots used by the retired microcode operations.

3.3 Examples

3.3.1 Linux and 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 <application>
```

- 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  
<application>
```

- 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
```

3.3.2 Windows

Core Metrics

- 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:

```
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 the 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 channel:

```
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 the 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 XML file is available in `<uprof-install-dir>\bin\Data\Config\SamplePcm-core.conf`. This file can be copied and modified to certain 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
```

3.4 BIOS Settings - Known Behavior

Following is the known behavior of L2 Hit/Miss from HWPF metrics based on the BIOS settings:

- AMDuProfPcm L2 Hit/Miss from HWPF metric doesn't collect any data when all following options are disabled in BIOS:
 - L1 Stream HW Prefetcher
 - L1 Stride Prefetcher
 - L1 Region Prefetcher
 - L2 Stream HW Prefetcher
 - L2 up/Down Prefetcher

- AMDuProfPcm L2 Hit/Miss from HWPF metric collects very less samples with the following BIOS settings:
 - L1 Stream HW Prefetcher: Disable
 - L1 Stride Prefetcher: Disable
 - L1 Region Prefetcher: Enable
 - L2 Stream HW Prefetcher: Disable
 - L2 up/Down Prefetcher: Disable

3.5 Pre-release Features

This section describes the pre-release features in AMDuProfPcm tool in this release.

3.5.1 Monitoring without Root Privileges

On Linux, use the option `-x` to monitor the metrics without having a dependency on the "msr" module and root access. This option collects Core, L3, and DF PMC events on AMD “Zen”-based processors. The newer processors may require the latest kernel support.

Examples

- Timeseries monitoring of IPC of a benchmark, aggregate metrics per thread:


```
$ AMDuProfPcm -X -m ipc -o /tmp/pcm.csv -- /tmp/myapp.exe
```
- Timeseries monitoring of IPC of a benchmark, aggregate metrics per processor package:


```
$ AMDuProfPcm -X -m ipc -A package -o /tmp/pcm.csv -- /tmp/myapp.exe
```
- Timeseries monitoring of IPC of a benchmark, aggregate metrics at system level:


```
$ AMDuProfPcm -X -m ipc -A system -o /tmp/pcm.csv -- /tmp/myapp.exe
```
- Cumulative reporting of IPC metrics at the end of the benchmark execution:


```
$ AMDuProfPcm -X -m ipc -C -o /tmp/pcm.csv -- /tmp/myapp.exe
```
- Cumulative reporting of IPC metrics at the end of the benchmark execution, aggregate metrics per processor package:


```
$ AMDuProfPcm -X -m ipc -C -A package -o /tmp/pcm.csv -- /tmp/myapp.exe
```
- Cumulative reporting of IPC metrics at the end of the benchmark execution, aggregate metrics at system level:


```
$ AMDuProfPcm -X -m ipc -C -A system -o /tmp/pcm.csv -- /tmp/myapp.exe
```
- Timeseries monitoring of memory bandwidth reporting at package and memory channels level:


```
$ AMDuProfPcm -X -m memory -a -A system,package -o /tmp/mem.csv
```
- Timeseries monitoring of level-1 and level-2 top-down metrics (pipeline utilization):


```
$ AMDuProfPcm -X -m topdown -A system -o /tmp/td.csv -- /tmp/myapp.exe
```
- Cumulative reporting of level-1 and level-2 top-down metrics (pipeline utilization):


```
$ AMDuProfPcm -X -m topdown -C -A system -o /tmp/td.csv -- /tmp/myapp.exe
```

For better top-down results, disable NMI watchdog and run the following command as root:

```
echo 0 > /proc/sys/kernel/nmi_watchdog
```

3.5.2 Roofline Model

AMDuProfPcm provides basic roofline modeling that relates the application performance to memory traffic and floating point computational peaks. This is a visual performance model offering insights on improving the parallel software for floating point operations. This helps to characterize an application and identify whether a benchmark is memory or compute bound.

The tool monitors the memory traffic and floating point operations when the profiled application is running. Also, it computes the Arithmetic Intensity that is “operations per byte of DRAM traffic [FLOPS/BYTE]”. The roofline chart is plotted as:

- X-axis: (AI) Arithmetic Intensity (FLOPS/byte) in logarithmic scale
- Y-axis: Throughput (GFLOPS/sec) in logarithmic scale
- Horizontal line showing peak theoretical floating-point performance of the system (HW Limit).
- Diagonal line showing peak memory performance. This line is plotted using the formula $\text{Throughput} = \min(\text{peak theoretical GFLOPS/Second}, \text{Peak theoretical Memory Bandwidth} * \text{AI})$.

By default, the tool plots horizontal rooflines for:

- Single Precision Floating Point Peak ("SP FP Peak")
- Double Precision Floating Point Peak ("DP FP Peak")

The options available to plot the max peak horizontal (computational) peak rooflines are:

- Single precision noSIMD and noFMA
- Double precision noSIMD and noFMA

Generating the roofline chart of an application:

1. Collect the profile data using AMDuProfPcm:

```
$ AMDuProfPcm roofline -X -o /tmp/myapp-roofline.csv -- /tmp/myapp.exe
```

On AMD “Zen4” 9xx4 Series processors, if the Linux kernel doesn't support accessing DF counters, use the following command with root privilege:

```
$ AMDuProfPcm roofline -o /tmp/myapp-roofline.csv -- /tmp/myapp.exe
```

2. To generate the roofline chart, run the following command:

```
$ AMDuProfModelling.py -i /tmp/myapp-roofline.csv -o /tmp/ --memspeed 3200 -a myapp
```

The roofline chart is saved in the file */tmp/AMDuProf_roofline-2022-10-28-19h00m10s.pdf*.

A few pointers for generating the roofline chart:

- While collecting the data, if the AMDuProfPcm is launched with non-root privilege, specify the DRAM speed using `-memspeed` option. You can use `dmidecode` or `lshw` command to get the memory speed.
- To plot additional computational horizontal peaks line, use the following options:
 - `--sp-roofs`: Plot maximum peak roof for single-precision noSIMD and noFMA
 - `--dp-roofs`: Plot maximum peak roof for double-precision noSIMD and noFMA

Example:

```
$ AMDuProfModelling.py -i /tmp/myapp-roofline.csv -o /tmp/ --memspeed 3200 -a myapp -dp-roofs
```

- Use `-a <appname>` option to specify the application name to print in the graph chart.
- As this tool uses the maximum theoretical peaks for memory traffic and floating-point performance, you can use benchmarks such as STREAM to get the peak memory bandwidth and HPL or GEMM for peak FLOPS. Those scores can be used to plot the roofline charts. Use the following options:
 - `--stream <STREAM score>`
 - `--hpl <HPL score>`
 - `--gemm <SGEMM | DGEMM score>`

A sample roofline chart is as follows:

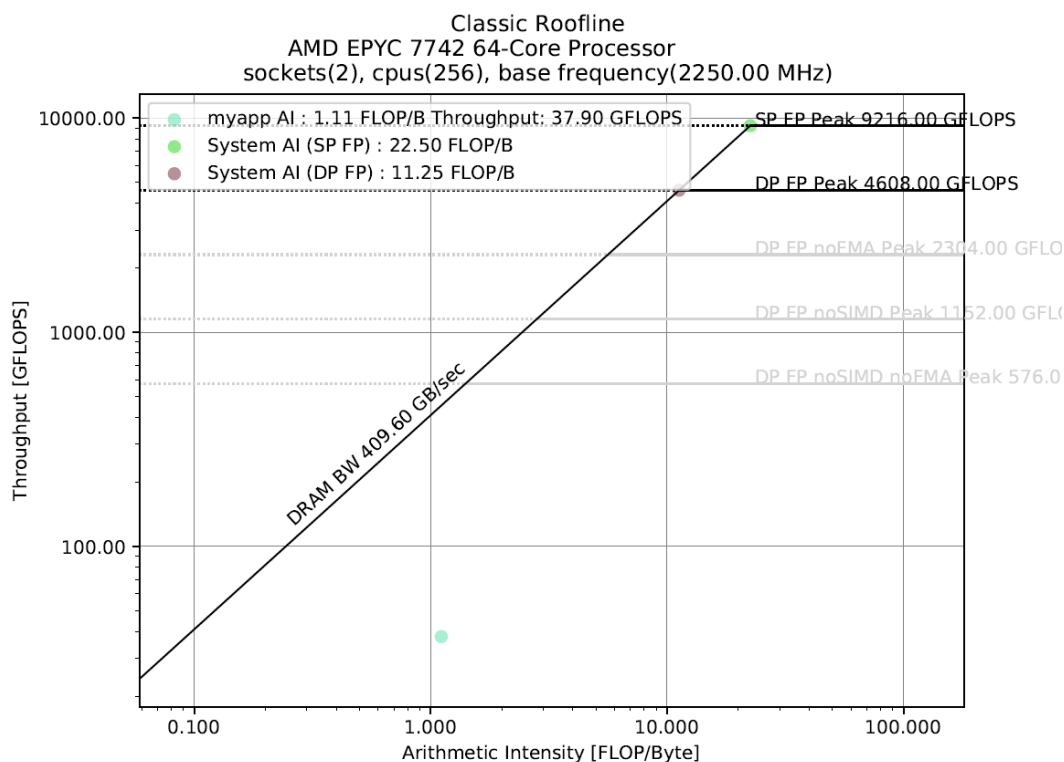


Figure 1. Sample Roofline Chart

3.5.3 Pipeline Utilization (Top-down Analysis)

On AMD “Zen4”-based processors, AMDuProfPcm supports monitoring and reporting the pipeline utilization (top-down) metrics. This feature provides top-down metrics to visualize the bottlenecks in the CPU pipeline. Use the option `-m topdown` to monitor and report the level-1 and level-2 top-down metrics.

The level-1 metrics are as follows:

Table 18. Level-1 Metrics

Metric	Description
Total_Dispatch_Slots	Total dispatch slots; up to six instructions can be dispatched in one cycle.
SMT_Dispatch_contention	Unused dispatch slots as the other thread was selected.
Frontend_Bound	Dispatch slots that remained unused because the frontend did not supply appropriate instructions/ops.
Bad_Speculation	Dispatched operations that did not retire.
Backend_Bound	Dispatch slots that remained unused because of backend stalls.
Retiring	Dispatch slots used by operations that retired.

The level-2 metrics are as follows:

Table 19. Level-2 Metrics

Metric	Description
Frontend_Bound.Latency	Unused dispatch slots due to latency bottleneck in the frontend, such as Instruction Cache or ITLB misses.
Frontend_Bound.BW	Unused dispatch slots due to bandwidth bottleneck in the frontend, such as decode bandwidth or Op Cache fetch bandwidth.
Bad_Speculation.Mispredicts	Dispatched operations that were flushed due to branch mispredicts.
Bad_Speculation.Pipeline_Restarts	Dispatched operations that were flushed due to pipeline restarts (resyncs).
Backend_Bound.Memory	Dispatched slots that remained unused because of stalls due to memory subsystem.
Backend_Bound.CPU	Dispatched slots that remained unused because of stalls not related to the memory subsystem.
Retiring.Fastpath	Dispatch slots used by fastpath operations that retired.
Retiring.Microcode	Dispatch slots used by microcode operations that retired.

Due to multiplexing, the reported metrics may be inconsistent. To minimize the impact of multiplexing, use the option `-x`. For better results, use `taskset` to bind the monitored application to a specific set of cores and monitor only the cores on which the monitored application is running.

Run the following command to collect the top-down metrics:

```
$ sudo AMDuProfPcm -m topdown -c core=0 -A system -o /tmp/myapp-td.csv -- /usr/bin/taskset -c 0 myapp.exe
```

(or, use the pre-release option -X that does not require root access)

```
$ AMDuProfPcm -X -m topdown -A system -o /tmp/myapp-td.csv -- /usr/bin/taskset -c 0 myapp.exe
```

A sample report is as follows:

Total_Dispatch_Slots	SMT_Dispatch	Frontend_Bound	Bad_Speculation	Backend_Bound	Retiring	IPC	Frontend_Bound.Latency	Frontend_Bound.BW	Bad_Speculation.Mispredicts	Bad_Speculation.Backend_Bound.Memory	Backend_Bound.CPU	R
31528583352	0	0.18	0	33.19	66.39	3.85	0.18	0	0	13.15	20.04	
31801404234	0	0.19	0.96	32.79	66.98	3.95	0.19	0	0.91	0.05	13.13	19.66
31876659852	0	0.18	0.22	32.73	67.53	3.95	0.18	0	0.2	0.02	13	19.73
31889169876	0	0.18	0.08	32.54	66.96	4.01	0.19	0	0.08	0	12.94	19.6
31914934404	0	0.19	0	32.76	66.92	3.97	0.19	0	0	0	12.99	19.77
31991329650	0	0.19	0	39.84	59.33	3.53	0.21	0	0	0	14.25	25.59
31851469458	0	0.19	0.22	59.47	40.21	2.36	0.22	0	0.04	0.18	16.48	42.99
31949156628	0.01	0.19	0.21	59.3	40.39	2.33	0.22	0	0.04	0.17	16.43	42.87
31817293530	0	0.19	0.6	59.15	40.39	2.38	0.22	0	0.1	0.5	16.52	42.64
31832631192	0	5.95	18.46	43.48	32.79	1.84	5.44	0.5	18.39	0.07	20.61	22.87
31912889772	0	10	31.77	31.46	26.5	1.48	9.21	0.79	31.75	0.02	19.36	12.1
31883125182	0	10.08	31.78	31.71	26.5	1.47	9.24	0.84	31.76	0.02	19.51	12.2
31993782744	0	10	30.81	31.72	26.47	1.45	9.15	0.85	30.79	0.02	19.37	12.35
31858516134	0	10	31.56	31.83	26.32	1.45	9.15	0.85	31.54	0.02	19.6	12.23
31928600622	0	9.95	31.78	31.85	26.49	1.43	9.03	0.92	31.76	0.02	19.62	12.24
31802799444	0	9.98	32.24	31.71	26.48	1.45	9.13	0.85	32.22	0.02	19.66	12.05
31827460368	0	5.4	17.59	38.25	40.03	2.25	5.04	0.37	17.56	0.03	21.73	16.52
31937807754	0	0.14	0.05	44.44	54.97	3.22	0.17	0	0.03	0.02	22.78	21.66
31865155098	0	0.15	0.07	44.79	54.97	3.22	0.17	0	0.04	0.03	22.97	21.83
31977071160	0	0.15	0	44.79	54.89	3.18	0.17	0	0	0	22.8	22

Figure 2. Sample Report

Examples

- Timeseries monitoring of level-1 and level-2 top-down metrics (pipeline utilization) of a single-threaded program:

```
# AMDuProfPcm -m topdown -c core=1 -o /tmp/td.csv -- /usr/bin/taskset -c 1 /tmp/myapp.exe
```

- Timeseries monitoring of level-1 and level-2 top-down metrics of a multi-threaded program running on all the cores:

```
# AMDuProfPcm -m topdown -a -A system -o /tmp/td.csv -- /tmp/myapp.exe
```

- Cumulative monitoring of level-1 and level-2 top-down metrics of a multi-threaded program running on all the cores:

```
# AMDuProfPcm -m topdown -a -A system -C -o /tmp/td.csv -- /tmp/myapp.exe
```

Chapter 4 Getting Started with AMDuProfSys – System Analysis

4.1 Overview

AMDuProfSys is a python-based system analysis tool for AMD processors. In Linux, by default, this tool is based on the underlying Linux perf utility provided by the Linux kernel, msr module, and basic hardware access primitives provided by the kernel. However, you can choose to use AMDuProfDriver to collect data using the `--use-amd-driver` option. In Windows, AMDuProfSys uses the AMDuProf driver to collect the PMU event counters. This tool can be used to collect the hardware events and evaluate the simple counter values or complex recipe using collected raw events.

Supported Platforms

AMDuProfSys supports AMD EPYC™ 7002, 7003, and 9000 Series processors with the following variants:

- Family 17, model 0x30 - 0x3F
- Family 19, model 0x0 - 0xF
- Family 19, model 0x1 - 0x1F
- Family 19, model 0x20 - 0x2F

Supported Hardware Counters

- CORE PMC
- DF PMC (only for Family17, Model 0x30 to 0x3F and Family19, Model 0x10 to 0x1F)
- L3 PMC
- UMC PMC (only for Family19, Model 0x10 to 0x1F)

Supported Operating Systems

- Linux
- Windows

The AMDuProfSys tool is available in the following installation directory:

<Installed Directory>/bin/AMDPerf/AMDuProfSys.py

Prerequisites

In Linux, user space tool for Linux perf module should be installed for core, l3, and df hardware counter access. If perf is not installed already, you can install it using following command on Ubuntu:

```
$ sudo apt-get install linux-tools-common linux-tools-generic linux-tools-`uname -r`
```


Optionally, you can also use AMDuProf driver to collect data and Linux perf is not required in that case. To install AMDuProf driver, refer to the section Installing Power Profiling Driver on Linux on page 23.

In Linux, NMI watchdog must be disabled, this requires root privileges:

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

AMDuProfSys requires the following modules of Python 3.x installed on the system:

- `tqdm` — use `pip3 install tqdm` to install
- `xlswriter` — use `pip3 install xlswriter` to install
- `yaml` — use `apt-get install python-yaml` or `pip3 install pyyaml` to install
- `yamlordereddictloader` — use `pip3 install yamlordereddictloader` to install

Synopsis

```
AMDuProfSys.py [<OPTIONS>] -- [<PROGRAM>] [<ARGS>]
```

<OPTIONS> — To collect, generate report, or get help for this tool

<PROGRAM> — Denotes a launch application to be profiled

<ARGS> — Denotes the list of arguments for the launch application

Common Usages

- Display help:

```
$ ./AMDuProfSys.py -h
```

- Collect core metrics for core 0 and run application on core:

```
./AMDuProfSys.py collect --config core -C 0 -o output taskset -c 0 <application>
```

- Generate the .csv format report from the session file generated during collection:

```
./AMDuProfSys.py report -i output_core.ses
```

- Generate report in .xls format:

```
./AMDuProfSys.py report -i output_core.ses -f xls
```

- Collect time series core metrics, at an interval of 1000 ms for core 0-5 and run application on core:

```
./AMDuProfSys.py collect --config core -C 0-5 -I 1000 -o output taskset -c 0 <application>
```

- Generate time series report in .csv format:

```
./AMDuProfSys.py report -i output_core.ses
```

- Collect metrics for core, L3, and DF metrics together:

```
./AMDuProfSys.py collect --config core,l3,df,umc -C 0-10 <application>
```

4.2 Options

4.2.1 Generic

The following table lists the generic options:

Table 20. AMDuProfSys Generic Options

Option	Description
-h, --help	Display the usage
-v, --version	Print the version
--system-info	System information
--enable-irperf	Enable irperf <i>Note: It is available only on Linux and requires root privilege.</i>
--mux-interval-core <ms>	Set the multiplexing interval in millisecond(s)
--mux-interval-l3 <ms>	Set the multiplexing interval in millisecond(s)
--mux-interval-df <ms>	Set the multiplexing interval in millisecond(s), the default MUX interval is 4 ms

4.2.2 Collect Command

The following table lists the collect command options:

Table 21. AMDuProfSys Collect Command Options

Option	Description
--config	To launch the given application and to monitor the raw events. Collect commands can be configured to use predefined set of config files or a single config file with its path.
-a, --all-cpus	Collect from all the cores. <i>Note: Options -c and -a cannot be used together.</i>
-C, --cpu <CPUs>	List of CPUs to monitor. Multiple CPUs can be provided as a comma separated list with no space: 0,1. Ranges of CPUs: 5-10.
-d, --duration <seconds>	Profile duration to run. <i>Note: It will not work if launch application is specified.</i>
-t, --tid <tid>	Monitor events on existing thread(s). Multiple TIDs can be provided as a comma separated list. <i>Note: It is available only on Linux.</i>
-p, --pid <pid>	Monitor events on existing process(es). Multiple PIDs can be provided as comma separated list. <i>Note: It is available only on Linux.</i>

Table 21. AMDuProfSys Collect Command Options

Option	Description
-o, --output <file>	Output file name to save the raw event count values.
--no-inherit	The child tasks will not be monitored. <i>Note: It is available only on Linux.</i>
-I, --interval <n>	Interval at which raw event count deltas will be stored in the file. This is a must for collecting time series data. <i>Note: It is available only on Linux.</i>
-V, --verbose	Print verbose.
-r --collect-raw <file>	Collect events using raw events file. Event files can be found in <i>bin/AMDPProf/data/0x19_0x1/raw_config/</i> directory. The report can be generated only at AMD. This option helps collect bigger set of metrics. <i>Note: This option cannot be used when --use-amd-driver flag is used in Linux.</i>
--use-amd-driver	This option can be used in Linux to collect counter data using AMDuProf driver instead of Linux perf.

4.2.3 Report Command

To generate a profile report with computed metrics. The collect command generates a profile session file with *.ses* extension and a raw counter data file for each type of profile collection. To generate the report, you must provide the session file with *-i* option as shown in the following command options:

Table 22. AMDuProfSys Report Command Options

Option	Description
-i, --input-file <file>	Input the session file generated by collect command.
--config <file>	Config file or options core, df, and l3 for event sets and metrics.
-o, --output <file>	Output file name in <i>.csv</i> or <i>.xls</i> format as configured.
-f, --format	Output file format in <i>.xls</i> or <i>.csv</i> . Default file format is <i>.csv</i> .
--group-by <system,package,numa,ccx>	Aggregate result based on group selected. Default is none.
-T, --time-series	Generate per core time series report. Only <i>.csv</i> format is supported. Must be collected with <i>-I</i> option to generate the time series data. <i>Note: It is available only on Linux.</i>
--set-precision <n>	Set floating point precision for reported metrics, the default value is 2.
-V, --verbose	Print verbose.

Creating Customized Config Profile

The AMDuProfSys requires three types of processor specific input files for profiling - Event, Metrics, and Config files. The AMDuProfSys package contains these files with the default settings.

To collect customized metrics, Event and Metrics files can be edited to accommodate the new events and metrics.

You can create a new custom Config file to configure only the required customized events and metrics. An example to use custom Config file:

```
./AMDuProfSys.py collect --config data/0xXX_0xYY/configs/core/core_custom_config.yaml -d 3
```

Where, XX and YY are the family and model of the supported AMD processors.

Details of those input files are as follows:

- Event Files

Event files are in *data\0xXX_YY\events\ folder*. These files define the respective hardware PMC counters with their unit mask and other respective attributes. These events are used in the metrics defined in the metrics file. Event files are in *.json* format and begin with the prefix of the counter type. For example, core related event file should begin with **core_**.

- Metrics Files

Metrics files are in *data\0xXX_YY\metrics\ folder*. Metrics are the recipe or formula for a meaningful parameter. For example, IPC is a metrics calculated by the formula:

$$ipc = ((retired_instructions))/((tsc)*(aperf)/(mperf))$$

Events used in the metrics must be declared in the corresponding event file. Metrics can be as simple as the raw pmc count (example, retired_instructions) or a complex metrics as IPC. Further, metrics can also be defined as hierarchy of another metrics. Metrics files are in *.json* format and begin with the prefix of the counter type. For example, core related event file should begin with **core_**.

- Config Files

Config files are in *data\0xXX_YY\configs\ folder*. The *config* file has three sections:

- **CPU** consists of the supported processors, various PMC hardware counters availability.
- **Events** are declared for configuring raw PMC events in each multiplexed group. Since, HW PMC counters are limited, multiplexing group must be configured.
- **Reports** generation using the metrics defined in the metrics file. Based on the list of metrics configured, report will be generated. Ensure that the required events for evaluating metrics are configured for collection in the event section. The config files are in *yaml* format and begin with the prefix of the counter type. For example, core related event file must begin with **core_**.

4.3 Examples

- Monitor the entire system to collect the metrics defined in config file and generate the profile report:

```
$ ./AMDuProfSys.py collect --config core -a sleep 5
```

- Launch the program with core affinity set to core 0 and monitor that core and generate profile report:

```
$ ./AMDuProfSys.py collect --config core -C 0 taskset -c 0 /tmp/scimark2
```

- Launch the program and monitor it to generate the profile report:

```
$ ./AMDuProfSys.py collect --config core -a /tmp/scimark2
```

Note: *-a or -C option is mandatory for multiplexing to work.*

- Collect and generate report in two steps:

- a. To generate a binary datafile `sci_perf.data` containing raw event count values:

```
$ ./AMDuProfSys.py collect --config data/0x17_0x3/configs/core/core_config.yaml -C 0 -o sci_perf taskset -c 0 scimark2
```

- b. To generate a report file `sci_perf.csv` containing computed metrics:

```
$ ./AMDuProfSys.py report -i sci_perf/sci_perf.ses -o sci_perf
```

- Collect using multiple config files and generate report in two steps:

- a. To generate a binary datafile `sci_perf.data` containing raw event count values:

```
$ ./AMDuProfSys.py collect --config core,l3,df -C 0-10 -o sci_perf taskset -c 0 scimark2
```

Note: *-C, -a option can be used only with the core counters.*

- b. To generate a report file `sci_perf.csv` containing computed metrics:

```
$ ./AMDuProfSys.py report -i sci_perf/sci_perf.ses -o all_events
```

- Update the multiplexing interval:

```
# ./AMDuProfSys.py --mux-interval-core 16
```

Note: *--mux-interval-core option requires root access.*

Chapter 5 Getting Started with AMD uProf GUI

5.1 User Interface

The AMD uProf GUI provides a visual interface to profile and analyze the performance data. It has various pages and each page has several sub-windows. You can navigate the pages through the top horizontal navigation bar. When a page is selected, its sub-windows will be listed in the leftmost vertical pane as follows:

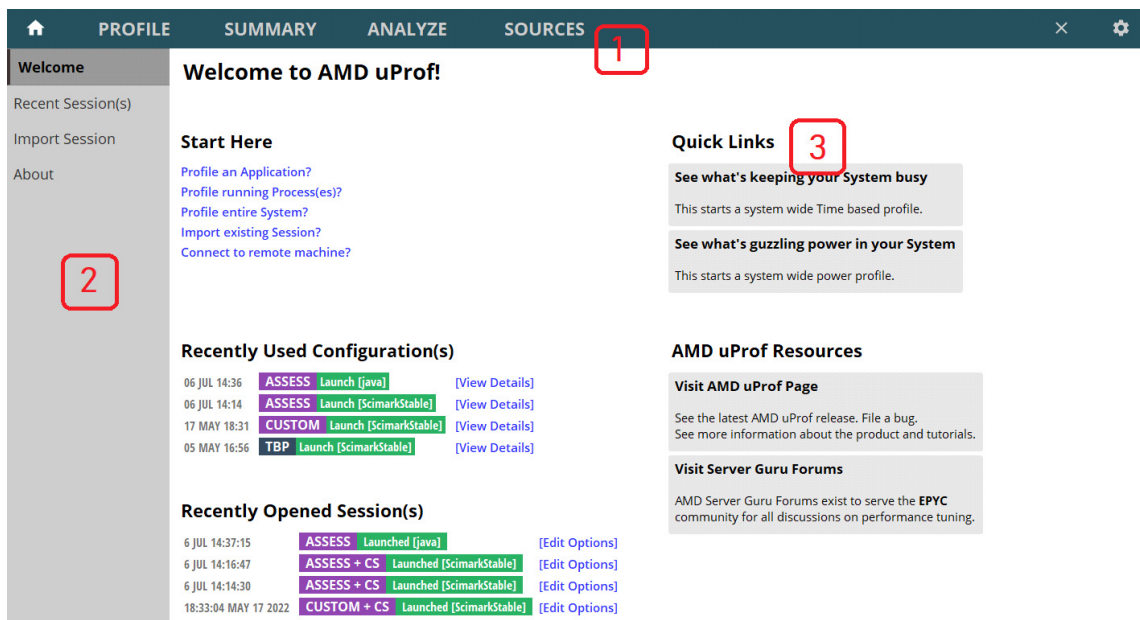


Figure 3. AMD uProf GUI

1. The menu names in the horizontal bar such as **HOME**, **PROFILE**, **SUMMARY**, and **ANALYZE** are called pages.
2. Each page has its sub-windows listed in the leftmost vertical pane. For example, **HOME** page has various windows such as **Welcome**, **Recent Session(s)**, **Import Session**, and so on.
3. Each window has various sections. These sections are used to specify various inputs required for a profile run, display the profile data for analysis, buttons and links to navigate to associated sections. In the **Welcome** window, **Quick Links** section has two links that allows you to start a profile session with minimal configuration steps.

5.2 Launching GUI

To launch the AMDuProf GUI program:

Windows

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

Linux

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

The *Welcome* screen is displayed as follows:

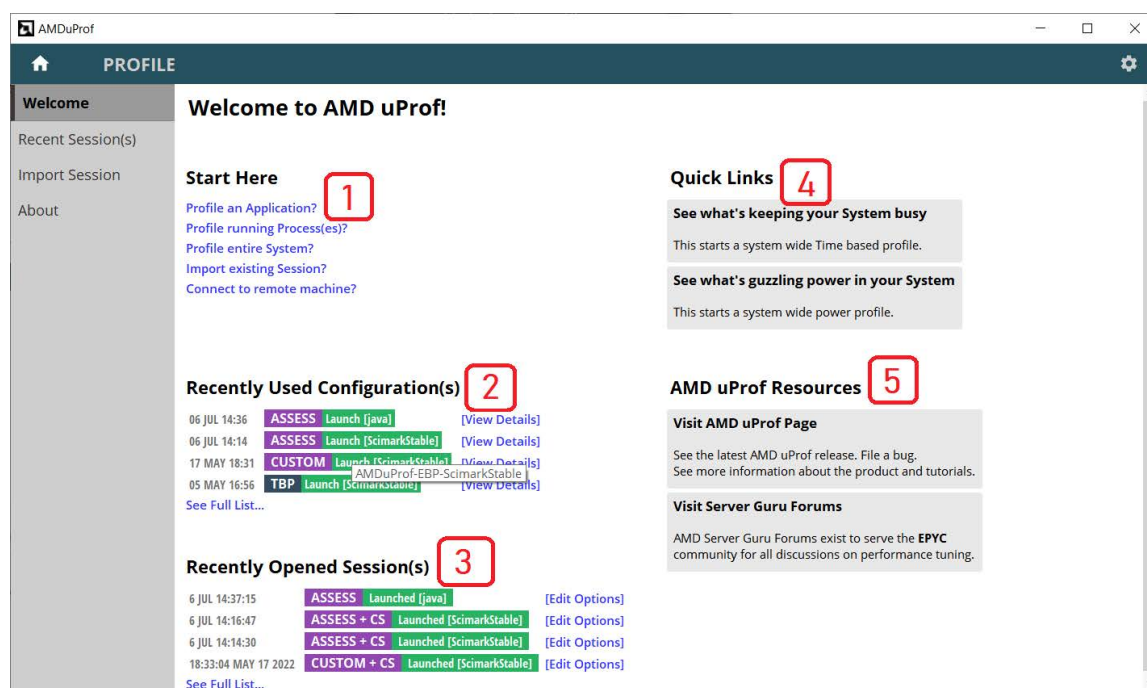


Figure 4. AMD uProf Welcome Screen

It has many sections as follows:

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. You 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. You 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.

- c. Click **See what's keeping your System busy** to start a system-wide time-based profiling until you stop it and then display the collected data.
 - d. Click **See what's guzzling power in your System** to select various power and thermal related counters and display a live view of the data through graphs.
5. **AMD uProf Resources** section provides links to the AMD uProf release page and AMD server community forum for discussions on profiling and performance tuning.

5.3 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 Profile or Live Power Profile)
 - b. Monitoring events - how the data should be collected
 - c. Additional profile data (if needed) - callstack samples, profile scheduling, and so on

This is called profile configuration “Profile Configuration” on page 27 that identifies all the information used to perform a collect measurement.

***Note:** The additional profile data to be collected depends on the selected profile type.*

5.3.1 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 **Welcome** screen. The **Start Profiling** screen is displayed. **Select Profile Target** is available in the **Start Profiling** window as follows:

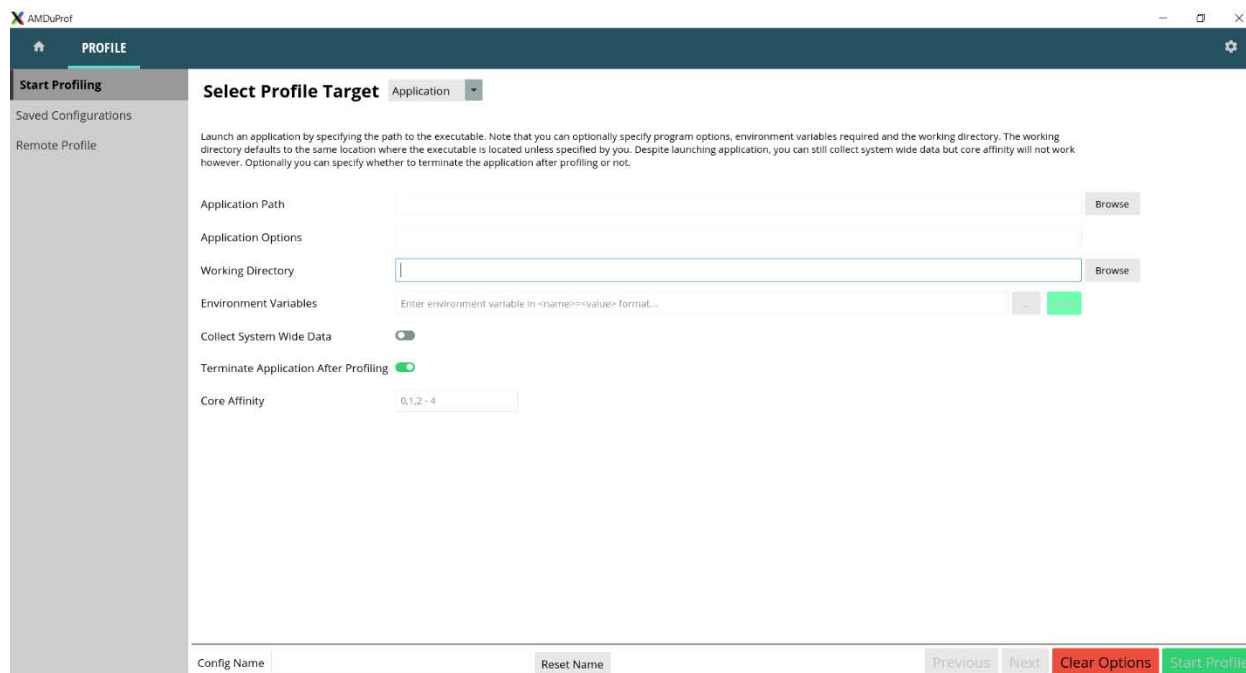


Figure 5. Start Profiling - Select Profile Target

You can select the one of the following profile targets from the **Select Profile Target** drop-down:

- **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 processes 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 screen of **Start Profiling**.

***Note:** The **Next** button will be enabled only if all the selected options are valid.*

5.3.2 Select Profile Type

Once profile target is selected and configured, click the **Next** button. The **Select Profile Type** screen is displayed as follows:

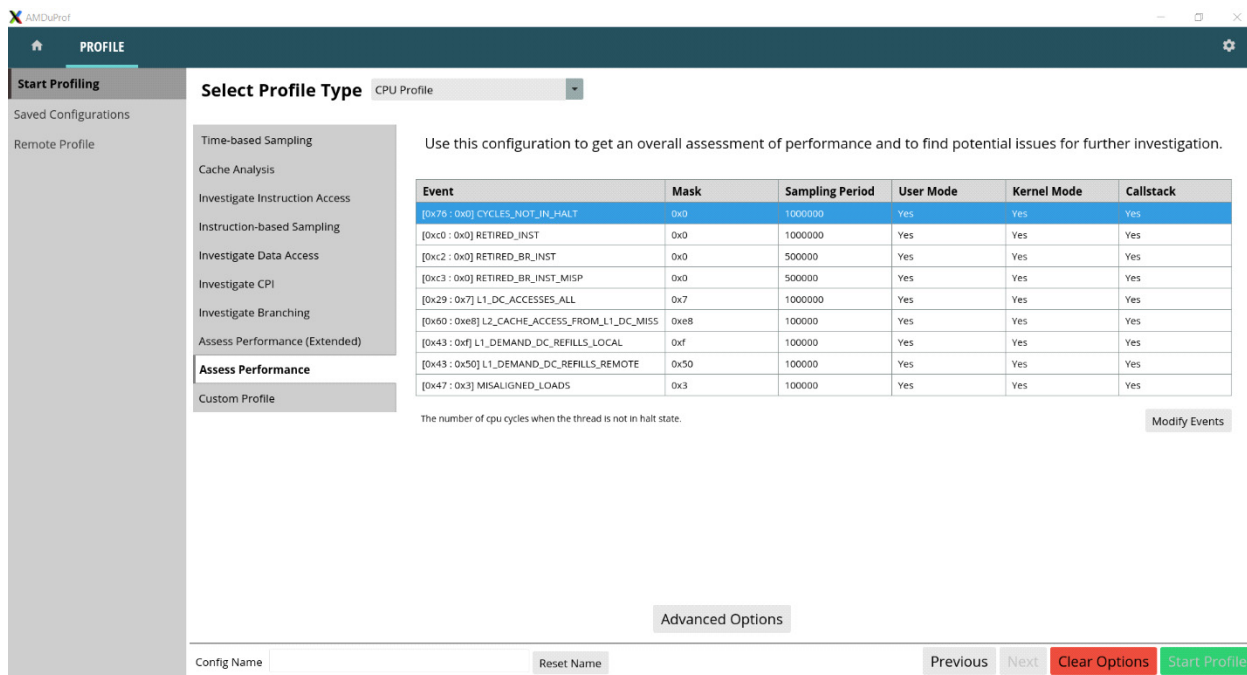


Figure 6. Start Profiling - Select Profile Type

This screen 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. In the above figure:

1. **Select Profile Type** drop-down 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. 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, and user mode) for that event. You can click **Modify Events** button to modify these event attributes and add new events and/or remove events.
4. Click **Advanced Options** button to proceed to the **Advanced Options** screen and set the other options such as the **Call Stack Options, Profile Scheduling, Sources, Symbols**, and so on.
5. The details in “Profile Configuration” on page 27 are persistent and saved by the tool with a name (here, it is *AMDuProf-EBP-ScimarkStable*). You can define this name and navigate to **PROFILE > Saved Configurations** to reuse/select the same configuration later.
6. The **Next** and **Previous** buttons are available to navigate to various screen of the **Start Profiling** screen.

5.3.3 Advanced Options

Click the **Advanced Options** button in **Select Profile Type** screen. The **Advanced Options** screen is displayed as follows:

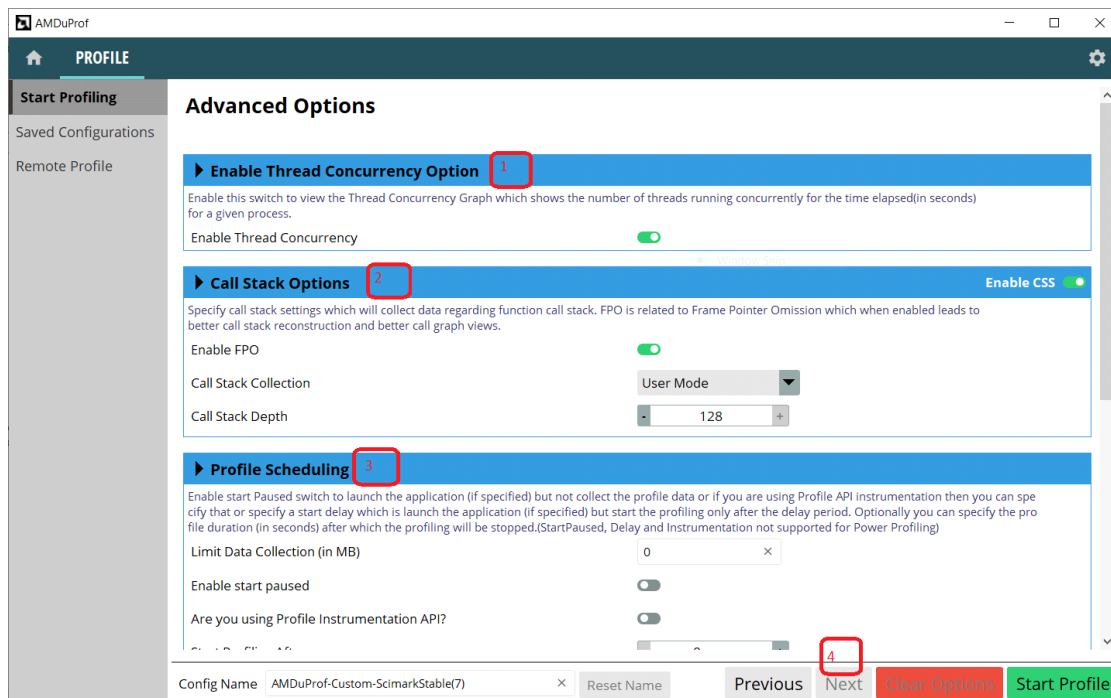


Figure 7. Start Profiling - Advanced Options 1

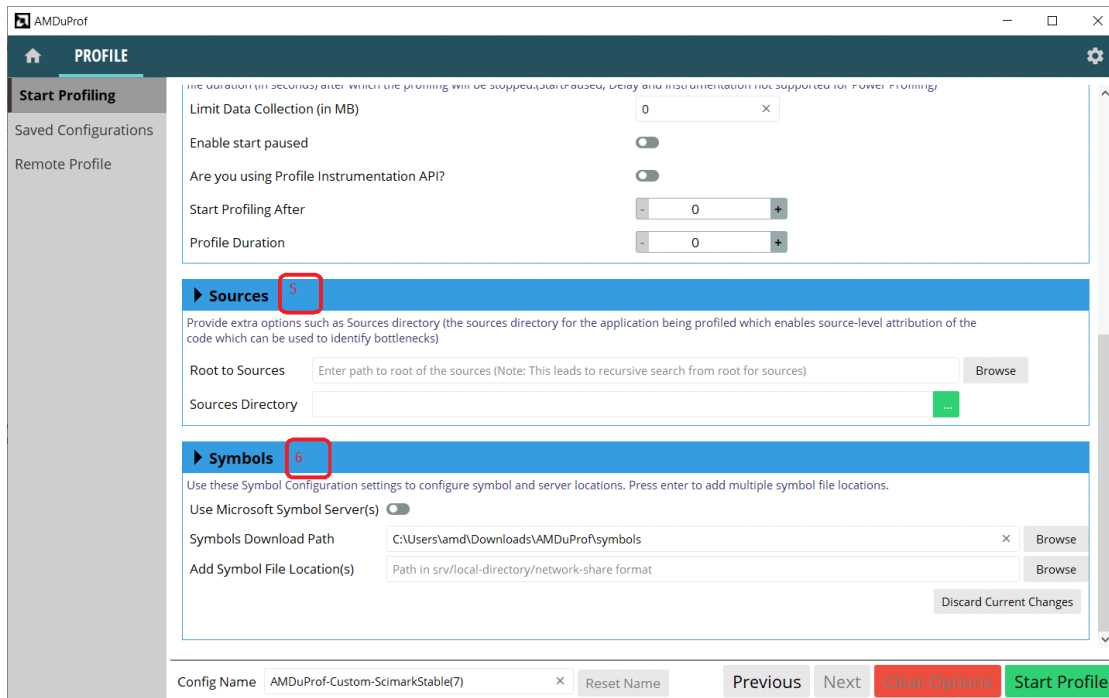


Figure 8. Start Profiling - Advanced Options 2

You can set the following options on the **Advanced Options** screen:

1. **Enable Thread Concurrency** to collect the profile data and to show Thread Concurrency Chart in Windows.
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** screen.
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.

5.3.4 Start Profile

Once all the options are set correctly, click the **Start Profile** button to start the profile and collect the profile data. After the profile initialization the following screen is displayed:

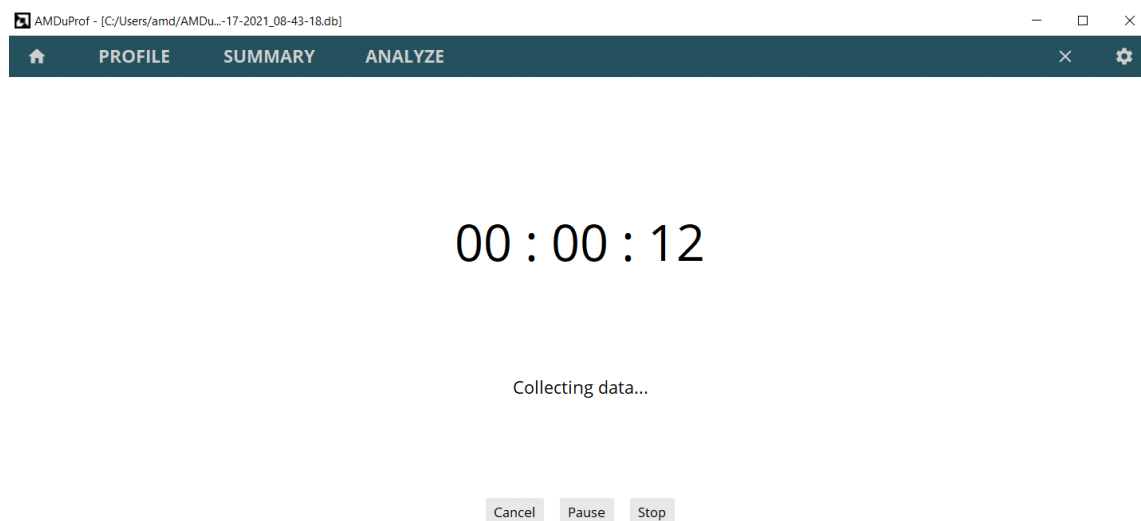


Figure 9. Profile Data Collection

1. The time elapsed during the data collection is displayed.
2. When the profiling is in progress, you can:
 - Click the **Stop** button to stop the profiling.
 - Clicking **Cancel** button to cancel the profiling. It will take you back to **Select Profile Target** screen of **PROFILE**.
 - Click the **Pause** button to pause the profiling. The profile data will not be collected and you can click the **Resume** button to continue the profiling.

5.4 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.

- **TIMECHART** page to visualize the MPI API trace, OS event trace, and GPU trace information as a timeline chart.

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

5.4.1 Overview of Performance Hotspots

When the translation is complete, the **SUMMARY** page will be populated with the profile data and **Hot Spots** screen will be displayed. The **SUMMARY** page provides an overview of the hotspots for the profile session through various screens such as **Hot Spots** and **Session Information**.

In the **Hot Spots** screen, hotspots will be displayed for functions, modules, process, and threads. Processes and threads will be displayed only if there are more than one.

The following figure shows the **Hot Spots** screen:

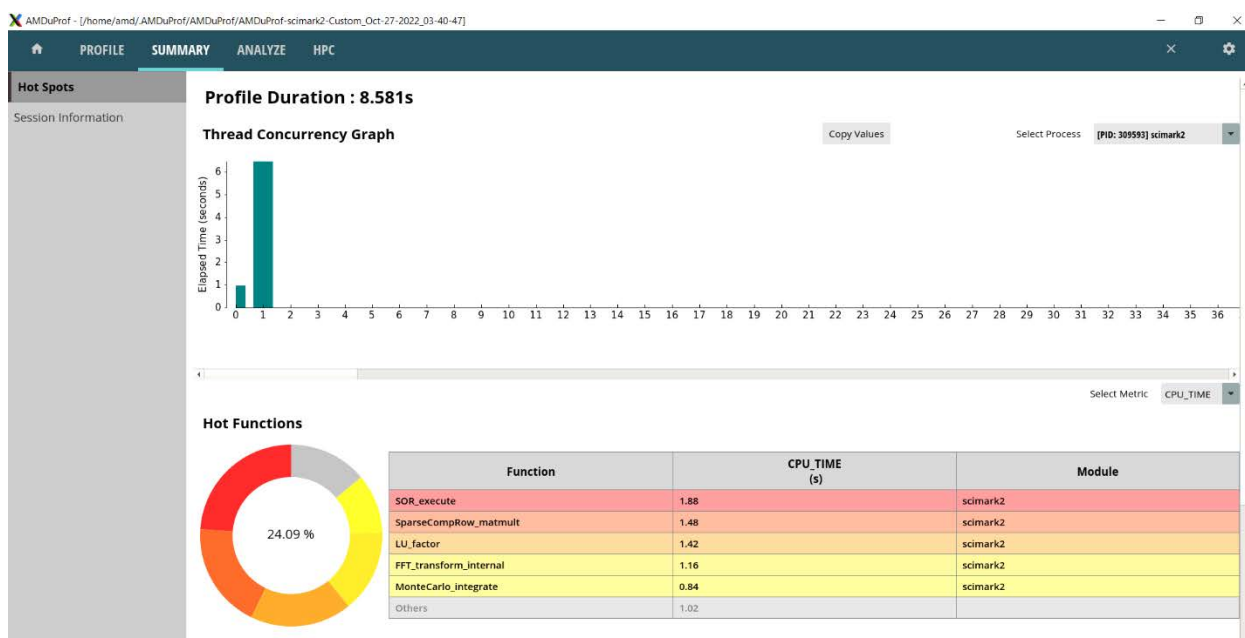


Figure 10. Summary - Hot Spots Screen

In the above **Hot Spots** screen:

1. The top 5 hottest functions, processes, modules and threads for the selected event are displayed.
2. The **Hot Functions** pie chart is interactive in nature. You can click on any section and the corresponding function's source will open in a separate tab in the **SOURCES** page.
3. The hotspots are shown per event and the monitored event can be selected from drop-down in the top-right corner. You can change it to any other event to update the corresponding hotspot data.

5.4.2 Thread Concurrency Graph

Click **ANALYZE** > **Thread Concurrency** to view the following graph to analyze the thread concurrency of the profiled application:

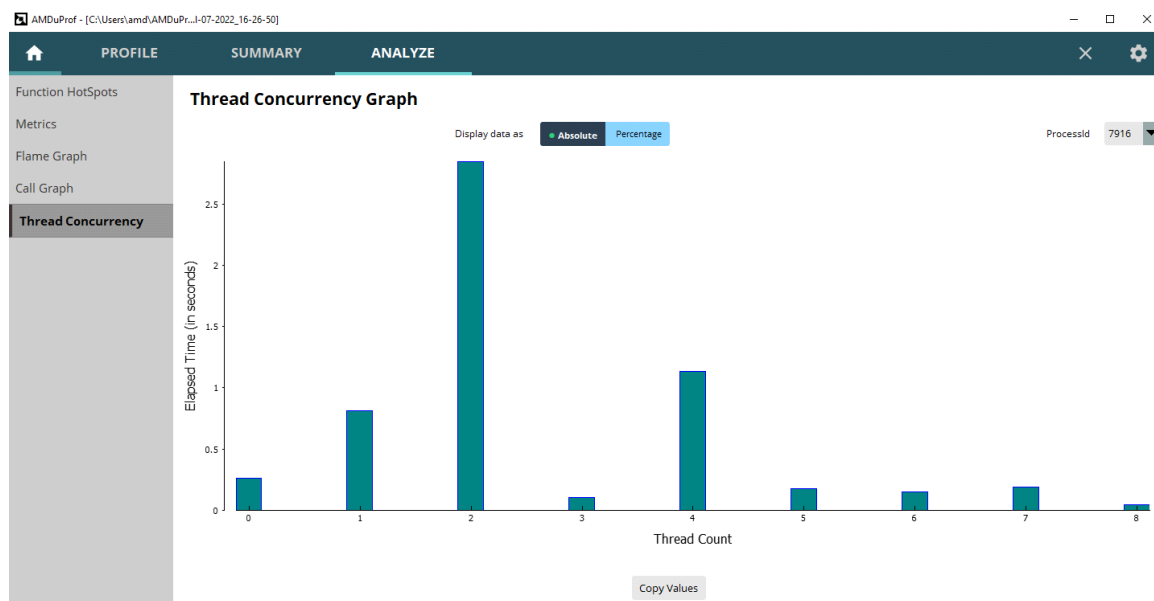


Figure 11. Summary - Thread Concurrency Graph

The thread concurrency graph displays the duration (in seconds) of the specific number of threads that were running simultaneously.

5.4.3 Function HotSpots

Click **ANALYZE** on the top horizontal navigation bar to go to **Function HotSpots** screen, which displays the hot functions across all the profiled processes and load modules as follows:

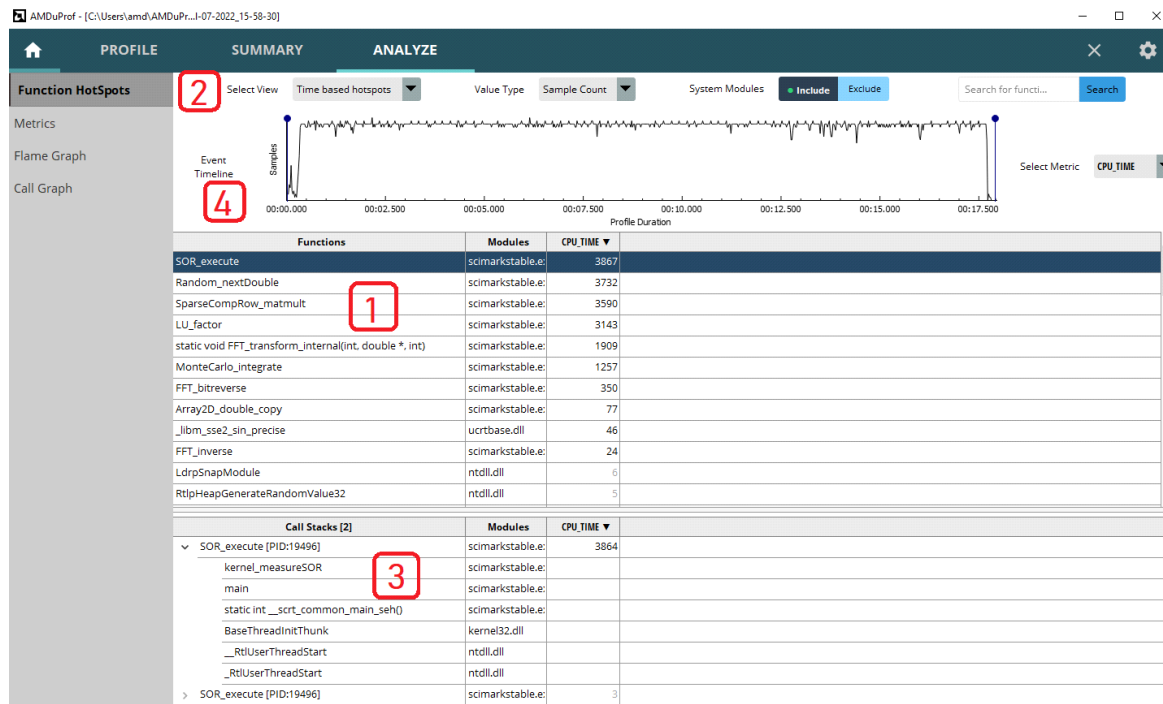


Figure 12. ANALYZE - Function HotSpots

Function HotSpots screen contains the following:

- The **Functions** table lists the hot functions. The IP samples are aggregated and attributed at the function-level granularity. On the table, you can do the following:
 - Double-click on a function entry to navigate to the corresponding **SOURCE** view of that function.
 - Right-click to view the following options:
 - Copy selected row(s)** to copy the highlighted row to clipboard.
 - Copy all rows** to copy all the rows to clipboard.
- Filters and Options** pane allows you filter the profile data as follows:
 - You can click the **Select View** drop-down to control the counters that are displayed. The relevant counters and their derived metrics are grouped in predefined views.
 - You can use the **Value Type** drop-down to display the counter values as follows:
 - 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.
 - You can use the **System Modules** option to either **Exclude** or **Include** the profile data attributed to system modules.

3. If callstack is enabled, the unique hot call-paths for the selected function is displayed in the **Functions** column.
4. **Event Timeline** is the line graph showing the number of aggregated sample values over the period of time. You can use it to identify the hot functions within a profile region. From the **Select Metric** drop-down you can select the event for which event timeline must be plotted.

All the entries will not be loaded for a profile. To load more than the default number of entries, click the vertical scroll bar on the right.

5.4.4 Process and Functions

Click **ANALYZE > Metrics** to display the profile data table at various program unit granularities - Process, Load Modules, Threads, and Functions. This screen contains data in two different formats as follows:

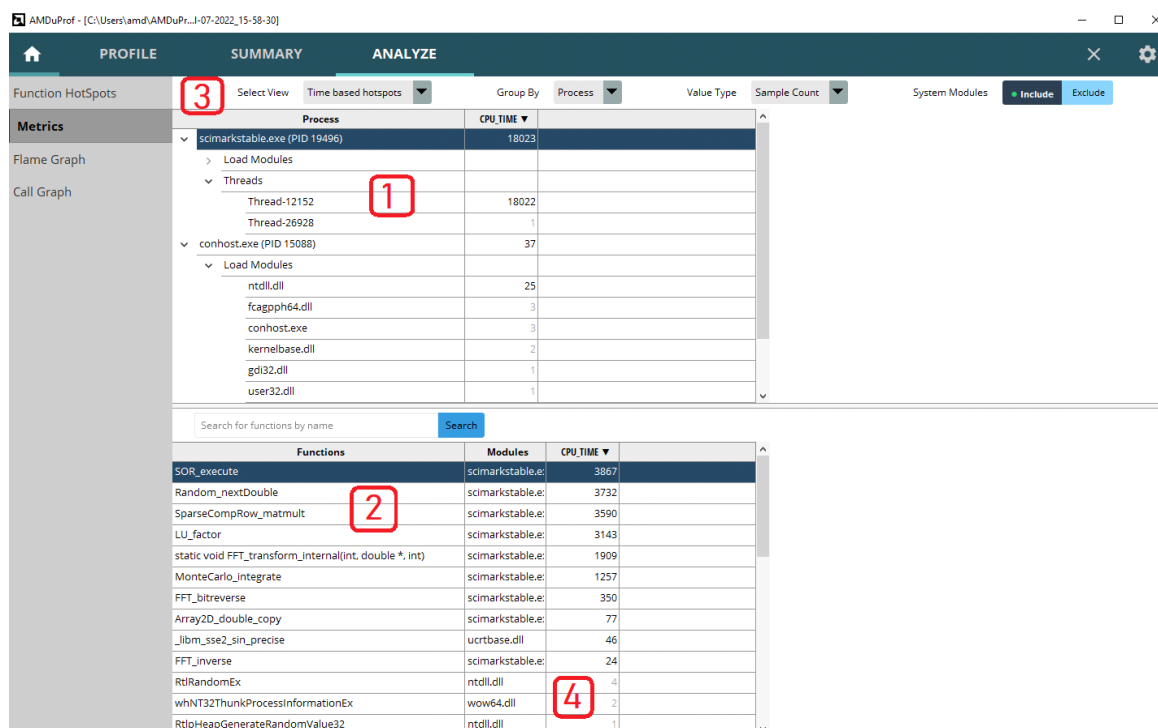


Figure 13. Analyze - Metrics

The above figure consists of the following:

1. The upper tree represents samples grouped by **Process**. You can expand the tree to view the child entries for each parent (that is for a process). The **Load Modules** and **Threads** are child entries for the selected process entry.

You can right-click to view the following options:

- **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 row(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.- You can do any of the following:
 - Double-click on a function entry to navigate to the corresponding **SOURCE** view.
 - Right-click to view the following options:
 - **Copy selected row(s)** to copy the highlighted row to clipboard.
 - **Copy all rows** to copy all the rows to clipboard.
 3. You can use the **Filters and Options** pane to filter the profile data displayed by various controls.
 - The **Select View** controls the counters that are displayed. The relevant counters and their derived metrics are grouped in predefined views. You can select the views from the **Select View** drop-down.
 - The **Group By** drop-down is used to group the data by Process, Module, and Thread. By default, the sample data is grouped-by Process.
 - Click the **ValueType** drop-down to display the counter values as follows:
 - **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.
 - You can use the **System Modules** option to **Exclude** or **Include** the profile data attributed to system modules.
 4. Confidence level — The metrics that cannot be calculated reliably due to low number of samples collected for a program unit will be grayed out.

All entries will not be loaded for a profile. To load more than the default number of entries, click the vertical scroll bar on the right.

5.4.5 Source and Assembly

Double-click on any entry in the **Functions** table in the **Metrics** screen to load the source tab for the corresponding 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.

The following figure depicts the source and assembly screen:

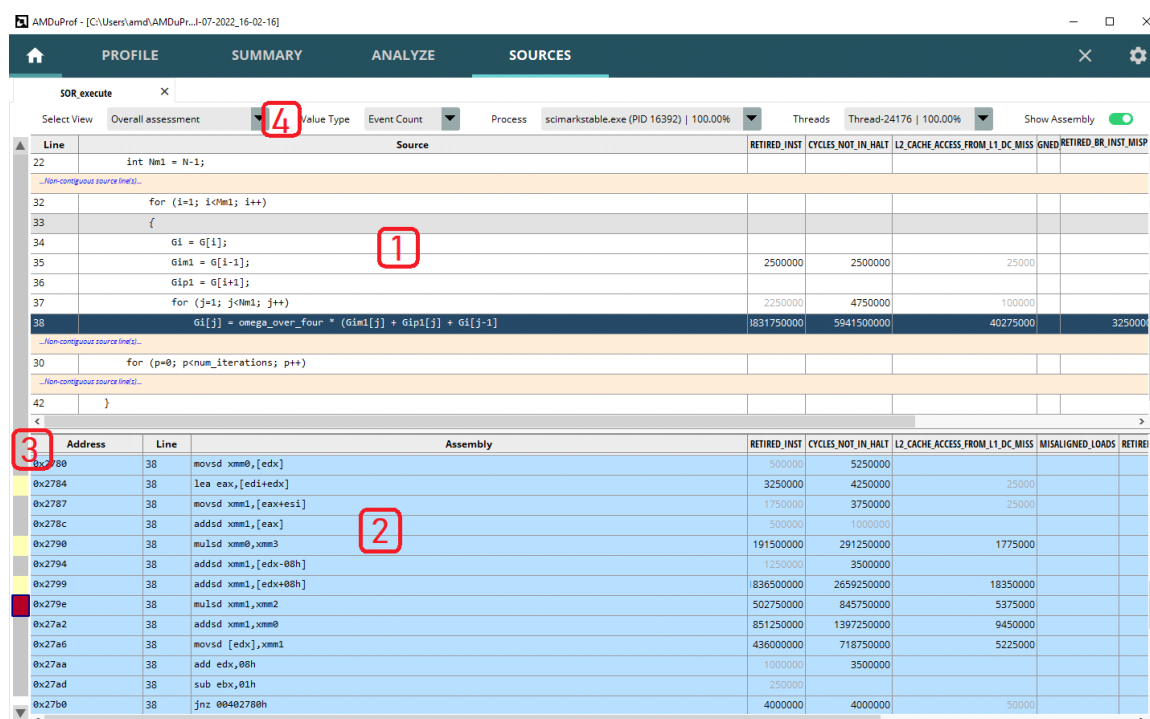


Figure 14. SOURCES - Source and Assembly

Following section are present in the **SOURCES** screen:

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.

4. **Filters** pane lets you filter the profile data by providing the following options.

- The **Select View** controls the counters that are displayed. The relevant counters and their derived metrics are grouped in predefined views. You can select it from the **Select View** drop-down.
- The **Process** drop-down lists all the processes on which this selected function is executed and has samples.
- The **Threads** drop-down lists all the threads on which this selected function is executed and has samples.
- You can use the **ValueType** drop-down to display the counter values as follows:
 - **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 **Show Assembly** button shows/hides visibility of the assembly instruction table shown at the bottom of the view.

For multi-threaded or multi-process applications, if a function has been executed from multiple threads or processes, each of them will be listed in the **Process** and **Threads** drop-downs in the **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 displayed.

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

5.4.6 Flame Graph

Flame graph is a visualization of sampled call-stack traces to quickly identify the hottest code execution paths. Click **ANALYZE > Flame Graph** to view it as follows:

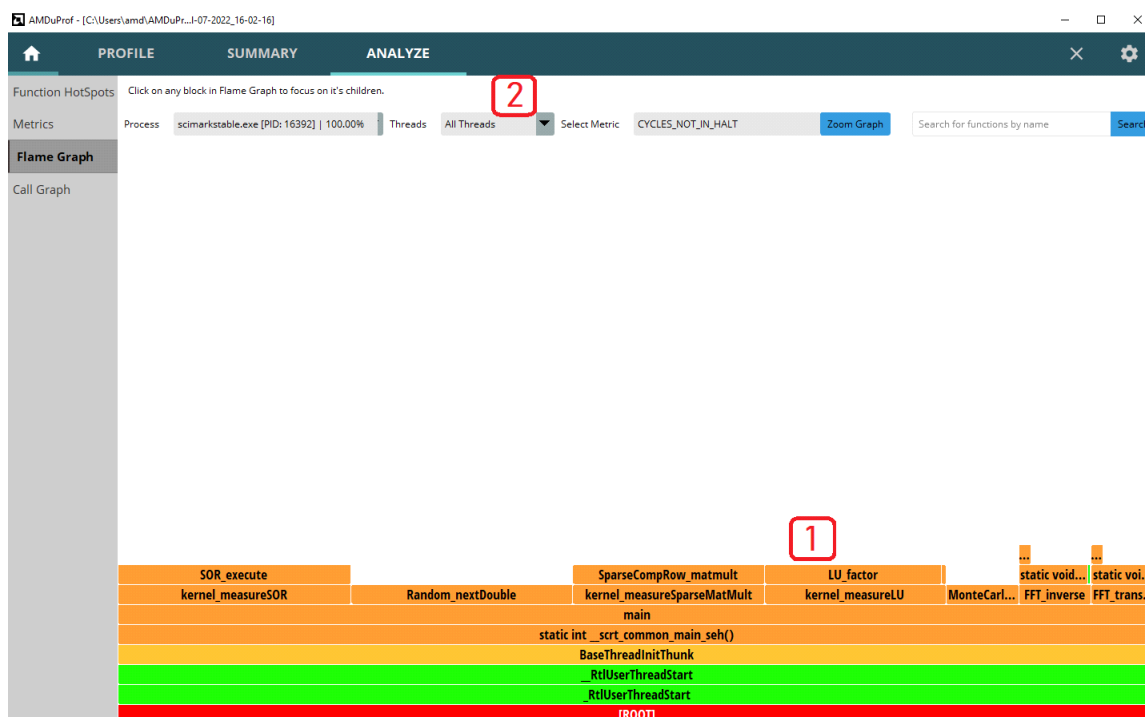


Figure 15. ANALYZE - Flame Graph

The **Flame Graph** screen comprises of the following:

1. The x-axis of the flame graph shows the call-stack 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 call-stack samples, the cell would be wider. This screen has the following options:
 - Module-wise coloring of the cells.
 - Click on a cell to zoom only that cell and its children. Use the **Reset Zoom** button visualize the entire graph.
 - Right-click on a cell to view the following context options:
 - **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.
 - Hover the mouse over a cell to display the tool-tip showing the inclusive and exclusive number of samples of that function.

2. Following options are available at the top of this screen:

- Click **Zoom Graph** button for a better zooming experience.
- When you type a function name in the search box, a list of all the relevant matches will be displayed. Select the required function to highlight the cells corresponding to that function in the flame graph.
- The **Process** drop-down lists all the processes for which call-stack samples are collected. Changing the process will plot the flame graph for that particular process.
- For multi-threaded applications, the flame graph will be plotted for the cumulative data of all the threads by default.
- The **Threads** drop-down lists all the threads for which call-stack samples are collected. Changing the thread will plot the flame graph for that thread.
- The **Select Metric** drop-down lists all the metrics for which call-stack samples are collected. Changing the metric will plot the flame graph for that particular metric.

5.4.7 Call Graph

Click **ANALYZE > Call Graph** to navigate to the call graph screen. This graph is constructed using the call-stack samples and offers a butterfly view to analyze the hot call-paths as follows:

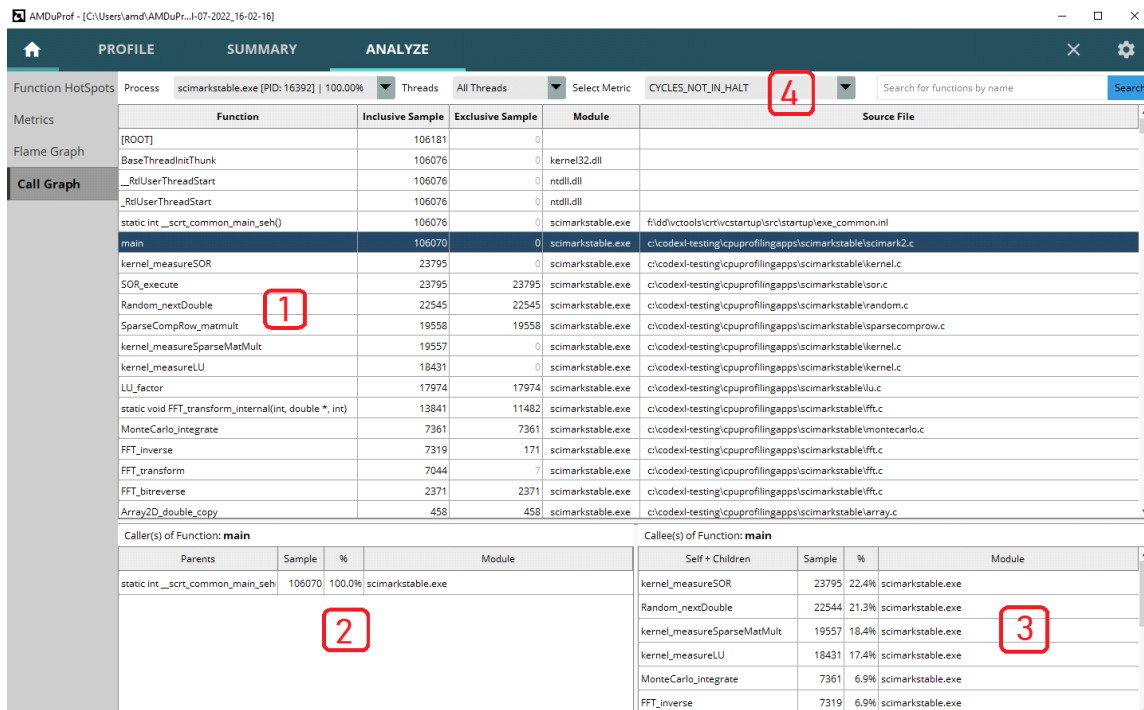


Figure 16. ANALYZE - Call Graph

1. The Function table lists all the functions with inclusive and exclusive samples.
Click on function to display its Caller and Callee functions in a butterfly view.
2. Lists all the parents of the function selected in the Function table.
3. Lists all the children of the function selected in the Function table.
4. Options:
 - The **Process** drop-down lists all the processes for which call-stack samples are collected. Changing the process will show the call graph for that particular process.
 - For multi-threaded applications, the call-graph will be plotted for the the cumulative data of all the threads by default.
 - The **Threads** drop-down lists all the threads for which call-stack samples are collected. Changing the thread will plot the call graph for that thread.
 - The **Select Metric** drop-down lists the metrics for which call-stack samples are collected. Changing the counter will show the call graph for that particular counter.

5.4.8 IMIX View

IMIX view shows the summary of instruction-wise samples collected. This view is shown only for IBS profiling. Click **ANALYZE > IMIX** to navigate to the IMIX view:

Function HotSpots	Select Metric	Module	Functions
Metrics	IBS_TAG_TO_RET	scimarktable.exe	FFT_bitreverse
IMIX	Instruction	Sample Count	Percentage
	cmp	30256	0.28
	mov	25096	0.23
	lea	13266	0.12
	movups	8536	0.08
	sub	8451	0.08
	sar	7608	0.07
	inc	6761	0.06
	add	6758	0.06
	movsxd	6298	0.06
	nop	4829	0.04
	jnl	150	0.00
	cdq	10	0.00
	retq	10	0.00

Figure 17. IMIX View

1. The IMIX table lists all the instructions with sample count and sample percentage for the selected options.

2. Options:

- The **Select Metric** drop-down lists all the metrics for which samples are collected. Changing the metric will display the IMIX information for that metric.
- The **Module** drop-down lists all the binaries for which samples are collected. Changing the module will display the IMIX information for that module.
- The **Functions** drop-down lists all the functions for which samples are collected. Changing the function will display the IMIX information for that thread. By default, IMIX information for All Functions is shown.

5.5 Importing Profile Database

To analyze a profile database generated using CLI, click **HOME > Import Session** to go to the **Import Profile Session**. The following screen is displayed:

Figure 18. Import Session – Importing Profile Database

This can be used to import the processed profile data collected using the CLI or the processed profile data saved in GUI's profile session storage path. You must do the following:

- Specify the path containing the *session.uprof* file in the **Profile Data File** box.
- **Binary Path:** If the profile run is performed in a system and the corresponding raw profile data is imported in another system, you must 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. No sub-directories will be searched in this path to locate any source files.
- **Root Path to Sources:** Specify the path to the root of multiple source directories. The entire directory and sub-directories present in that path will be searched to locate any source files.
Note: The search might take time as all the sub-directories will be searched recursively.
- **Force Database Regeneration:** To forcefully regenerate the database file while importing.

- **Use Cached Source/Binary/Symbol Files:** Enable this option to reuse cached source, binary, and symbol files.

5.6 Analyzing Saved Profile Session

Once you have created a new profile session or opened (imported) profile database, the history is updated and the last 50 opened profile database records are stored (that is, where they are located). Such a list will also appear in **HOME > Recent Session(s)** as follows:

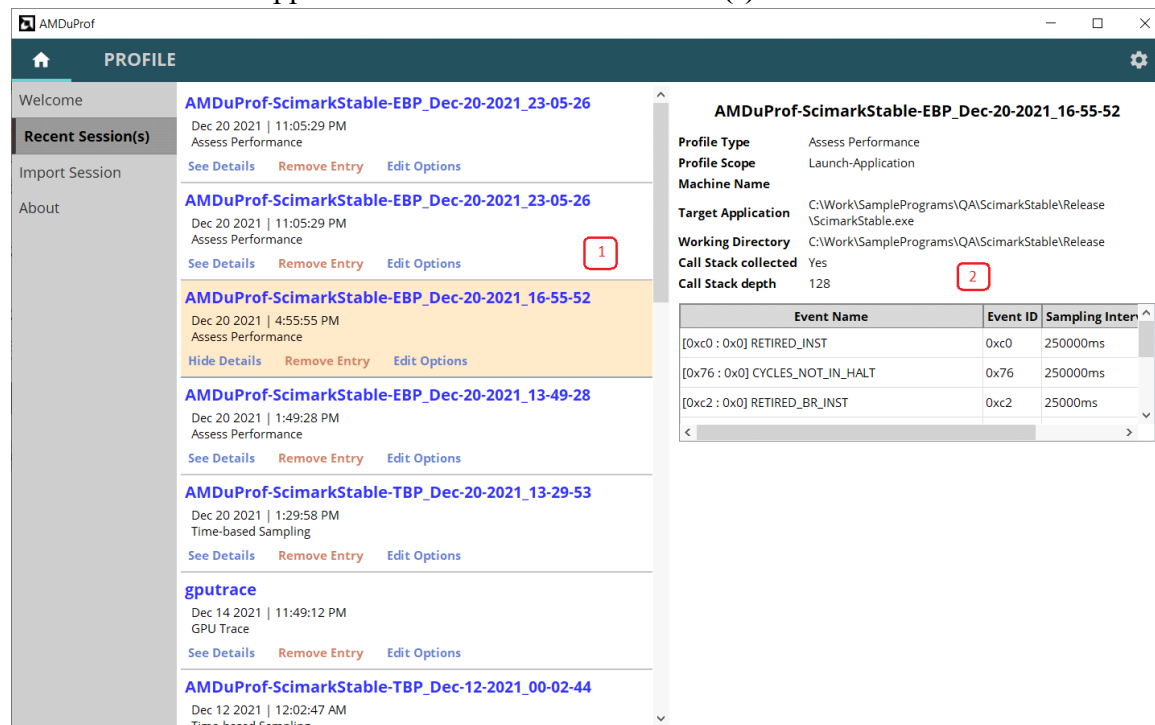


Figure 19. PROFILE - Recent Session(s)

In the above figure:

1. History of profile sessions opened for analysis in the GUI. The following options are available:
 - Click on an entry to load the corresponding profile database for analysis.
 - **See Details** button displays details about this profile session such as profiled application, monitored events list, and so on.
 - Click **Edit Options** to automatically fill the **Import Profile Session** for the database and update the required line-edits before opening the session.
 - **Remove Entry** button deletes the current profile session from the history.
2. Displays the details of the selected profile session.

5.7 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. Such a list is available in **PROFILE > Saved Configurations** as follows:

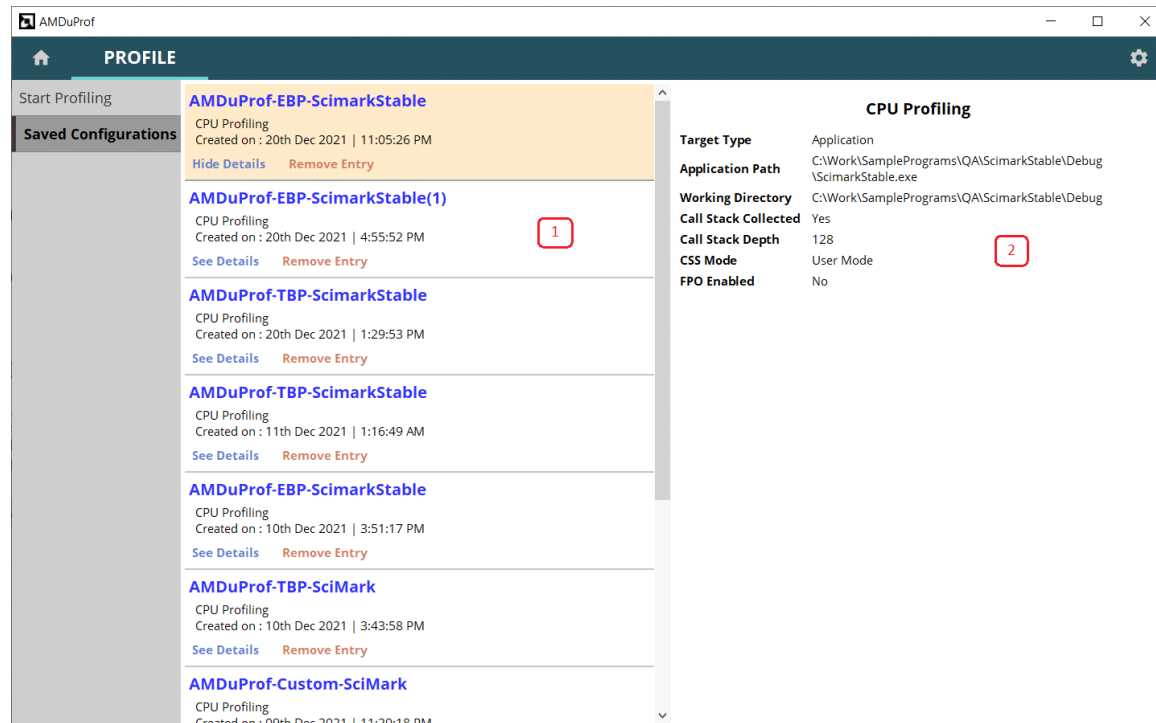


Figure 20. PROFILE - Saved Configurations

In the above figure:

- History of profile configurations used to collect profile data using GUI. The following options are available:
 - Click on an entry to display the corresponding profile configuration for data collection.
 - See Details** button displays the details about the current profile session such as profiled application, monitored events list, and so on.
 - Remove Entry** button deletes the current profile session from the history.
- Displays the details of the selected profile session.

Note: By default, the profile configuration name is generated by the application. If you want to reuse it, you should name it appropriately to locate it easily. To do so, provide a config name in the bottom left corner (**Config Name** line-edit) in **PROFILE > Start Profiling**.

5.8 Settings

There are certain application-wide settings to customize the AMD uProf experience. The **SETTINGS** page is in top-right corner and is divided into the following three sections:

- **Preferences:** Use this section to set the global path and data reporting preferences

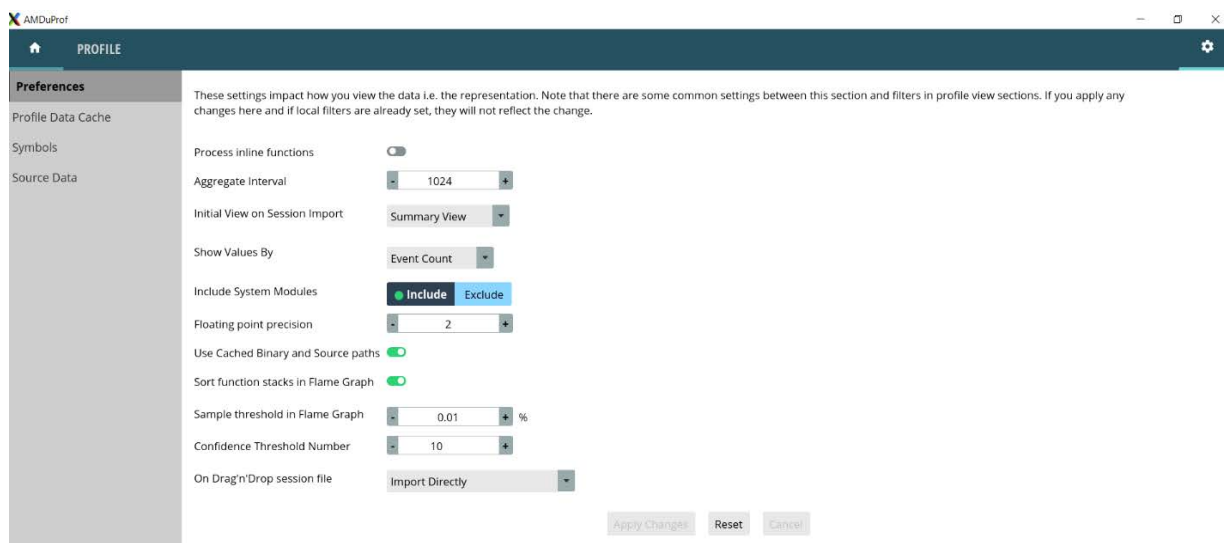


Figure 21. SETTINGS - Preferences

- Click the **Apply Changes** button to apply the updated/modified settings. There are settings which are common to profile data filters and hence, any changes to them through the **Apply Changes** button will only be applied to the views that do not have local filters set.
- You can click **Reset** button to reset the settings or **Cancel** to discard the 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. The following figure represents the **Symbols** section:

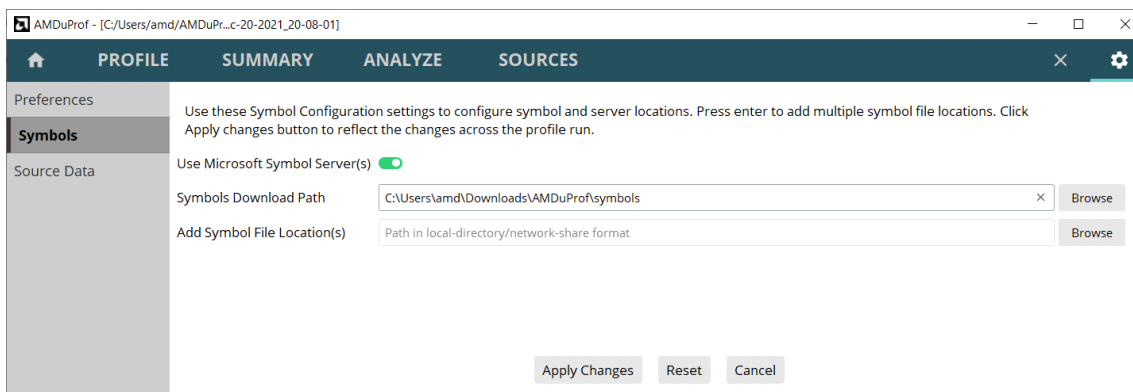


Figure 22. SETTINGS - Symbols

- **Source Data:** Use this section to set the Source view preferences. The following figure represents the **Source Data** section:

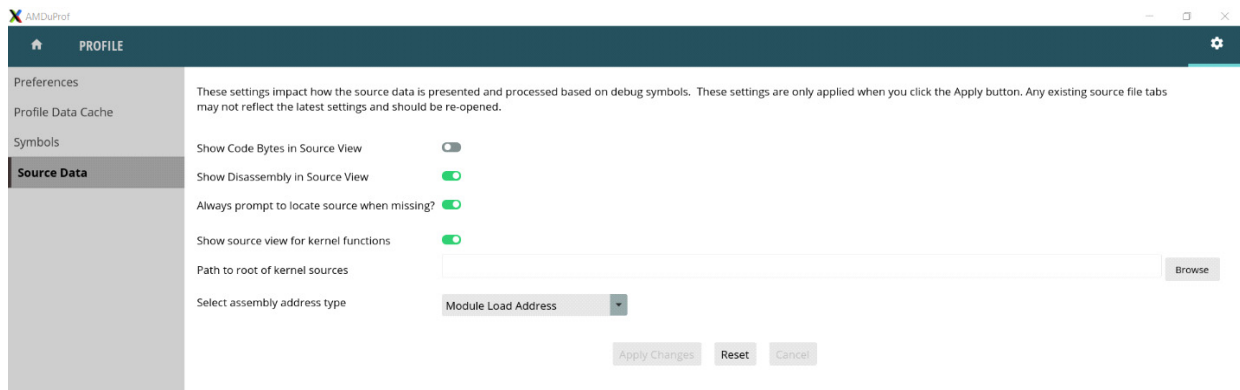


Figure 23. SETTINGS - Source Data

- **Profile Data Cache:** Use this section to control the location of data generation during profiling. The following figure represents Profile Data Cache section:

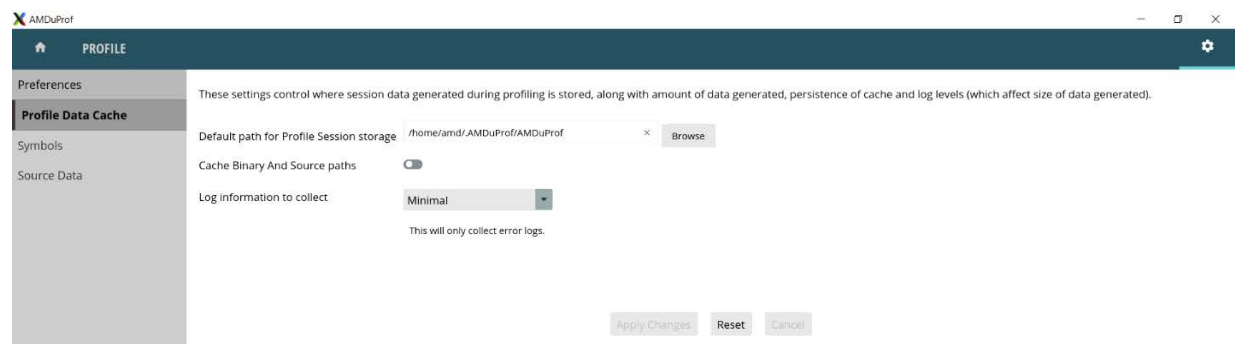


Figure 24. Profile Data Cache

Chapter 6 Getting Started with AMD uProf CLI

6.1 Overview

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>]
```

The following commands are supported:

Table 23. Supported Commands

Command	Description
collect	Runs the given program and collects the profile samples.
report	Processes the raw profile datafile and generates profile report.
timechart	Power Profiling — collects and reports system characteristics, such as power, thermal, and frequency metrics.
info	Displays the generic information about system and topology.
translate	Processes the raw profile datafile and generates the profile DB.

For more information on the workflow, refer “Workflow and Key Concepts” on page 26. To run the command line interface AMDuProfCLI, run the following binaries as per the OS:

- Windows

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

- Linux:

```
/opt/AMDuProf_X.Y-ZZZ/bin/AMDuProfCLI
```

If installed using the .tar file:

```
./AMDuProf_Linux_x64_X.Y.ZZZ/bin/AMDuProfCLI
```

- FreeBSD:

```
sh ./AMDuProf_FreeBSD_x64_X.Y.ZZZ/bin/AMDuProfCLI
```

6.2 Starting a CPU Profile

To profile and analyze the performance of a native (C/C++) application, you must complete the following steps:

1. Prepare the application. For more information on preparing an application for profiling, refer “Reference” on page 193.

2. Use AMDuProfCLI `collect` command to collect the samples for the application.

Note: Run AMD uProf on FreeBSD with `sudo` command or root privilege.

3. Using AMDuProfCLI `report` command to generate a report in readable format for analysis.

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

The `collect` command launches the application (if given) and collects the profile data for the given profile type and sampling configuration. It generates the raw data file (.prd on Windows, .pdata on FreeBSD, 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. Also, it writes them into a database and then generates a report in the CSV file format.

The following figure shows how to run time-based profile and generate a report for the launch application *AMDTClassicMatMul.exe*:

```
C:\Program Files\AMD\AMDuProf\bin>AMDuProfCLI.exe collect --config tbp -o c:\Temp\cpu-prof "c:\Program Files\AMD\AMDuProf\Examples\AMDTClassicMatMul\bin\AMDTClassicMatMul.exe"
Profiling started...

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

Invoke inefficient implementation of matrix multiplication
Elapsed time: 1.4400 sec (0.0010 sec resolution)
Profiling (data collection) completed.
Generated data files path: c:\Temp\cpu-prof\AMDuProf-AMDTClassicMatMul-TBP_Dec-20-2021_16-32-21

C:\Program Files\AMD\AMDuProf\bin>AMDuProfCLI.exe report -i c:\Temp\cpu-prof\AMDuProf-AMDTClassicMatMul-TBP_Dec-20-2021_16-32-21
Translation started ...
Translation finished
Generated database file : cpu
Report generation started...
Generating report file...

Report generation completed...

Generated report file: c:\Temp\cpu-prof\AMDuProf-AMDTClassicMatMul-TBP_Dec-20-2021_16-32-21\report.csv
C:\Program Files\AMD\AMDuProf\bin>
```

Figure 25. Collect and Report Commands

6.2.1 List of Predefined Sample Configurations

To get the list of supported predefined sampling configurations that can be used with `collect` command's `--config` option, run the following command:

```
AMDuProfCLI info --list collect-configs
```

A sample output is as follows:

```
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]

  energy        : Power Profile
                  Use this configuration to identify where programs are consuming power.

  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, PMCx029, PMCx060, 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, PMCx029, PMCx060, PMCx047,
                    PMCx024, PMCx037, PMCx043]

  assess        : Assess Performance
                  Use this configuration to get an overall assessment of performance and
                  to find potential issues for investigation.
                  [PMU Events: PMCx0C0, PMCx076, PMCx0C2, PMCx0C3, PMCx029, PMCx060, PMCx047]
```

Figure 26. List of Supported Predefined Configurations

6.2.2 Profile Report

The profile report (in CSV format) contains the following sections:

- EXECUTION — Information about the target launch application.
- PROFILE DETAILS — Details about the current session, such as profile type, scope, and sampling events.
- MONITORED EVENTS — List of the profiled events and the corresponding sampling intervals.
- 10 HOTTEST FUNCTIONS — List of the top 10 hot functions and the metrics attributed to them.
- 10 HOTTEST PROCESSES — List of the top 10 hot processes and the metrics attributed to them.
- 10 HOTTEST MODULES — List of the top 10 hot modules and the metrics attributed to them.
- 10 HOTTEST THREADS — List of the top 10 hot threads and the metrics attributed to them.

- **PROFILE REPORT FOR PROCESS** — The metrics attributed to the profiled process. This section is shown when `--detail` option used for report generation. It contains other sub-sections, such as:
 - **THREAD SUMMARY** — List of threads with metrics attributed to them.
 - **MODULE SUMMARY** — List of load modules which belong to the process with metrics attributed to them.
 - **FUNCTION SUMMARY** — List of functions that belong 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.

6.3 Starting a Power Profile

6.3.1 System-wide Power Profiling (Live)

To collect power profile counter values, complete the following steps:

1. Run the AMDuProfCLI `timechart` command with `--list` option to get the list of supported counter categories.
2. Use the AMDuProfCLI `timechart` command for specifying the required counters with `--event` option to collect and the report the required counters.

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>
```

Figure 27. Output of timechart `--list` Command

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

```

C:\Program Files\AMD\AMDuProf\bin>AMDuProfCLI.exe timechart -e Power,Frequency -o c:\Temp\power-prof "c:\Program Files\AMD\AMDuProf\
Examples\AMDTClassicMatMul\bin\AMDTClassicMatMul.exe"
Profiling started...

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

Invoke inefficient implementation of matrix multiplication
Elapsed time: 1.6530 sec (0.0010 sec resolution)

Profile finished
Generated data files path: c:\Temp\power-prof\AMDuProf-AMDTClassicMatMul-Timechart_Dec-20-2021_16-20-54
Live Profile Output file : c:\Temp\power-prof\AMDuProf-AMDTClassicMatMul-Timechart_Dec-20-2021_16-20-54\timechart.csv
C:\Program Files\AMD\AMDuProf\bin>

```

Figure 28. Execution of timechart

The above run collects the power 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 is launched and the data is collected till the application terminates.

6.4 Collect Command

The `collect` command collects the performance profile data and writes into the raw data files in the specified output directory. These files can then be analyzed using `AMDuProfCLI report` command or `AMDuProf GUI`.

Synopsis:

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

<PROGRAM> — Denotes the 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>]

```

6.4.1 Options

The following table lists the `collect` command options:

Table 24. AMDuProfCLI Collect Command Options

Option	Description
-h --help	Displays the help information on the console/terminal.
-o --output-dir <directory-path>	Base directory path in which collected data files will be saved. A new sub-directory will be created in this directory.
--config <config>	Predefined sampling configuration to be used to collect samples. Use the command <code>info --list collect-configs</code> to get the list of supported configs. Multiple occurrences of <code>--config</code> are allowed.

Table 24. AMDuProfCLI Collect Command Options

Option	Description
-e --event or <predefined-event>	<p>A predefined event can be directly be used with -e, --event which has predefined arguments.</p> <p>Alternatively, for providing more granular parameters, specify Timer, PMU, IBS event, or a predefined event with arguments in the form of comma separated key=value pairs. The supported keys are:</p> <ul style="list-style-type: none"> • event=<timer ibs-fetch ibs-op> or <PMU-event> or <predefined-event> • umask=<unit-mask> • user=<0 1> • os=<0 1> • cmask=<count-mask> • inv=<0 1> • interval=<sampling-interval> • ibsop-count-control=<0 1> (for ibs-op event) • loadstore (for ibs-op event, only on Windows platform) • opl3miss (for ibs-op event, supported only on AMD “Zen4” processors) • fetchl3miss (for ibs-fetch event, supported only on AMD “Zen4” processors) • call-graph <p>Notes:</p> <ol style="list-style-type: none"> 1. It is not required to provide umask with predefined event. 2. Use the dedicated option --call-graph to specify the arguments related to the call stack sample collection. <p>Argument details:</p> <ul style="list-style-type: none"> • user – Enable(1) or disable(0) user space samples collection • os - Enable(1) or disable(0) kernel space samples collection • interval – Sample collection interval. For timer, it is the time interval in milliseconds. For PMU and predefined events, it is the count of the event occurrences. For IBS FETCH, it is the fetch count. For IBS OP, it is the cycle count or the dispatch count. • op-count-control – Choose IBS OP sampling by cycle(0) count or dispatch(1) count. • loadstore – Enable only the IBS OP load/store samples collection, other IBS OP samples are not collected. • opl3miss – Enable IBS OP sample collection only when a l3 miss occurs, for example, '-e event=ibs-op,interval=100000,opl3miss' • fetchl3miss – Enable IBS FETCH sample collection only when a l3 miss occurs, for example, '-e event=ibs-fetch,interval=100000,fetchl3miss'

Table 24. AMDuProfCLI Collect Command Options

Option	Description
	<p>When these arguments are not passed, then the default values are:</p> <ul style="list-style-type: none"> • umask=0 • cmask=0 • user=1 • os=1 • inv=0 • ibsop-count-control=0 (for ibs-op event) • interval=1.0 ms for timer event • interval=250000 for ibs-fetch, ibs-op, pmu-event, or predefined-event <p>Use the following commands as required:</p> <ul style="list-style-type: none"> • info --list predefined-events for the list of supported predefined events • info --list pmu-events for the list of supported PMU-events <p>Multiple occurrences of --event (-e) are allowed.</p>
-p --pid <PID...>	Profile the existing processes by attaching to a running process. The process IDs are separated by comma.
-a --system-wide	System Wide Profile (SWP) If this flag is not set, then the command line tool will profile only the launched application or the Process IDs attached with -p option.
-c --cpu <core...>	Comma separated list of CPUs to profile. The ranges of CPUs can be specified with '-', for example, 0-3. <i>Note: On Windows, the selected cores should belong to only one processor group. For example, 0-63, 64-127, and so on.</i>
-d --duration <n>	Profile only for the specified duration n in seconds.
--interval <num>	Sampling interval for PMC events. <i>Note: This interval will override the sampling interval specified with individual events.</i>
--affinity <core...>	Set the core affinity of the launched application to be profiled. Comma separated list of core-ids. The ranges of the core-ids must be specified, for example, 0-3. The default affinity is all the available cores.
--no-inherit	Do not profile the children of the launched application (processes launched by the profiled application).
-b --terminate	Terminate the launched application after the 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, there is no impact.

Table 24. AMDuProfCLI Collect Command Options

Option	Description
--start-paused	Profiling paused indefinitely. The target application resumes the profiling using the profile control APIs. This option must be used only when the launched application is instrumented to control the profile data collection using the resume and pause APIs (defined in the “AMDProfileControl APIs” on page 190).
-w --working-dir <path>	Specify the working directory. The default is the current working directory.
--log-path <path-to-log-dir>	Specify the path where the log file should be created. If this option is not provided, the log file will be created either in path set by AMDUPROF_LOGDIR environment variable or \$TEMP path (Linux, FreeBSD) or %TEMP% path (on Windows) by default. The log file name will be of the format \$USER-AMDuProfCLI.log (on Linux, FreeBSD) or %USERNAME%-AMDuProfCLI.log (on Windows).
--enable-log	Enable additional logging with log file.
--enable-logts	Capture the timestamp of the log records. It should be used with --enable-log option.
--limit-size <n>	Stop the profiling when the collected data file size (in MBs) crosses the limit n.

6.4.2 Windows Specific Options

The following table lists Linux specific collect commands:

Table 25. AMDuProfCLI Collect Command – Windows Specific Options

Option	Description
--call-graph <I:D:S:F>	Enables Callstack Sampling. Specify the Unwind Interval (I) in milliseconds and Unwind Depth (D) value. Specify the Scope (S) by choosing one of the following: <ul style="list-style-type: none"> • user: Collect only for the user space code. • kernel: Collect only for the kernel space code. • all: Collect for the code executed in the user and kernel space code. Specify to collect missing frames due to Frame Pointer Omission (F) by compiler: <ul style="list-style-type: none"> • fpo: Collect missing callstack frames. • fp: Ignore missing callstack frames.
-g	Same as passing --call-graph 1:128:user:fp.
--thread <thread=concurrency>	Collects the runtime thread details
--data-buffer-count <size>	Size (number of pages per core) of the buffer used for data collection by the driver. The default size is 256 pages per core.

Table 25. AMDuProfCLI Collect Command – Windows Specific Options

Option	Description
--trace os	Trace the target domain OS. Support provided for "schedule event" only. Use the command 'info --list ostrace-events' for a list of OS trace events.

6.4.3 Linux Specific Options

The following table lists Linux specific collect commands:

Table 26. AMDuProfCLI Collect Command – Linux Specific Options

Option	Description
--call-graph <F:N>	<p>Enables Callstack sampling. Specify (F) to collect/ignore missing frames due to omission of frame pointers by compiler:</p> <ul style="list-style-type: none"> fpo: Collect missing callstack frames. fp: Ignore missing callstack frames. <p>When F = fpo, (N) specifies the max stack-size in bytes to collect per sample collection. Valid range of the 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.</p> <p><i>Note: Passing a large N value will generate a very large raw data file.</i></p> <p>When F = fp, the value for N is ignored and hence, there is no need to pass it.</p>
-g	Same as passing --call-graph fp
--tid <TID,...>	Profile existing threads by attaching to a running thread. The thread IDs are separated by comma.
--trace <TARGET>	<p>To trace a target domain. TARGET can be one or more of the following:</p> <ul style="list-style-type: none"> mpi[=<openmpi mpich>,<ltwt full>] <p>Provide MPI implementation type:</p> <p>'openmpi' for tracing OpenMPI library</p> <p>'mpich' for tracing MPICH and its derivative libraries, for example, Intel MPI</p> <p>Provide tracing scope:</p> <p>'ltwt' for light-weight tracing</p> <p>'full' for complete tracing</p> <p>'--trace mpi' defaults to '--trace mpi=mpich,full'</p> openmp — for tracing OpenMP application. This is same as the option --omp. os[=<event1,event2,...>] — provide event names and optional threshold with comma separated list. syscall and memtrace events will take the optional threshold value as <event:threshold>. Use the command info --list ostrace-events for a list of OS trace events. gpu[=<hip,hsa>] — provide the domain for GPU Tracing. By default, the domain is set to 'hip,hsa'.

Table 26. AMDuProfCLI Collect Command – Linux Specific Options

Option	Description
--buffer-size <size>	Number of pages to be allotted for OS trace buffer. Default value is 256 pages per core. Increase the pages to reduce the trace data loss. This option is only applicable to OS tracing (--trace os).
--max-threads <thread-count>	Maximum number of threads for OS tracing. The default value is 1024 for launched application and 32768 for System Wide Tracing (-a option). Increase this limit when the application thread count increases more than the default limit. Otherwise, the behavior is undefined. <ul style="list-style-type: none"> • Launch App - Valid range: 1 to 4096 • System wide - Valid range: 1 to 4194304
--func <module:function-pattern>	Specify functions to trace from the library, executable, or kernel: <ul style="list-style-type: none"> • Function-pattern can be a function name or partial name ending with '*' or only '*' to trace all the functions of a module. • Module can be a library or executable. To trace the kernel functions, replace the module with "kernel". <p><i>Note: It is recommended to provide the absolute/full path of a module. If not, the search will be performed on the default library paths and not on the current working directory.</i></p>
--exclude-func <module:function-pattern>	Specify functions to exclude from the library, executable, or kernel: <ul style="list-style-type: none"> • Function-pattern can be a function name or partial name ending with '*' or only '*' to trace all the functions of a module. • Module can be a library or executable. To trace the kernel functions, replace the module with "kernel". <p><i>Note: It is recommended to provide the absolute path of a module. If not, the search will be performed on the default library paths and not on the current working directory.</i></p>
-m --mmap-pages <size>	Set the kernel memory mapped data buffer to size. The size can be specified in pages or with a suffix Bytes (B/b), Kilo bytes (K/k), Megabytes (M/m), and Gigabytes (G/g).
--omp	Profile OpenMP application. This option is the same as '--trace openmp'. <p><i>Notes:</i></p> <ol style="list-style-type: none"> 1. Applicable to process profiling. 2. Not applicable to System wide and Java app profiling. 3. Compile the OpenMP application with LLVM/Clang 8.0 or later. The supported base languages are C, C++, and Fortran.
--mpi	Pass this option while collecting CPU Profiling data of a MPI application. For MPI tracing, check the --trace option.
--kvm-guest <pid>	Specify the PID of qemu-kvm process to be profiled to collect guest-side performance profile.
--guest-kallsyms <path>	Specify the path of guest /proc/kallsyms copied on the local host. AMD uProf reads it to get the guest kernel symbols.

Table 26. AMDuProfCLI Collect Command – Linux Specific Options

Option	Description
--guest-modules <path>	Specify the path of guest /proc/modules copied to the local host. AMD uProf reads it to get the guest kernel module information.
--guest-search-path <path>	Specify the path of guest vmlinux and kernel sources copied on the local host. AMD uProf reads it to resolve the guest kernel module information.

6.4.4 Examples

Windows

- Launch application *AMDTClassicMatMul.exe* and collect the 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
$ ./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 the application *AMDTClassicMatMul.exe* and collect the 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 the IBS samples in the SWP mode:

```
C:\> AMDuProfCLI.exe collect --config ibs -a -o c:\Temp\cpuprof-ibs-swp AMDTClassicMatMul.exe
```

- Collect the 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 the 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 the samples for IBS OP with an interval of 50000:

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

- Launch *AMDTClassicMatMul.exe* and do TBP samples profile for thread concurrency, name:

```
C:\> AMDuProfCLI.exe collect --config tbp --thread thread=concurrency,name -o c:\Temp\cpuprof-tbp AMDTClassicMatMul.exe
```

- Launch *AMDTClassicMatMul.exe* and collect the Power samples in SWP mode:

```
C:\> AMDuProfCLI.exe collect --config energy -a -o c:\Temp\pwrprof-swp AMDTClassicMatMul.exe
```

- Collect samples for PMCx076 and PMCx0C0, but collect the call graph info only for PMCx0C0:

```
C:\> AMDuProfCLI.exe collect -e event=pmcx76,interval=250000 -e event=pmcx0c0,interval=250000,call-graph -o c:\Temp\cpuprof-pmc AMDTClassicMatMul-bin
```

- Launch *AMDTClassicMatMul.exe* and collect the samples for predefined event RETIRED_INST and L1_DC_REFILLS.ALL events:

```
C:\> AMDuProfCLI.exe collect -e event=RETIRED_INST,interval=250000 -e event=L1_DC_REFILLS.ALL,user=1,os=0,interval=250000 -o c:\Temp\cpuprof-pmc AMDTClassicMatMul.exe
```

- Launch *AMDTClassicMatMul.exe*, collect the TBP and Assess Performance samples:

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

- Launch *Mutithread_Threadname.exe* and collect schedule event:

```
C:\> AMDuProfCLI.exe collect --trace os -o c:\Temp\ost-output Multithread_Threadname.exe
```

Linux

- Launch application *AMDTClassicMatMul.bin* and collect the 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
$ ./AMDuProfCLI 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 the 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 the IBS samples in the SWP mode:

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

- Collect the 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 the 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 the samples for IBS OP with interval 50000:

```
$ ./AMDuProfCLI collect -e event=ibs-op,interval=50000 -o /tmp/cpuprof-tbp AMDTClassicMatMul-bin
```

- Attach to a thread and collect TBP samples for 10 seconds:

```
$ AMDuProfCLI collect --config tbp -o /tmp/cpuprof-tbp-attach -d 10 --tid <TID>
```

- Collect OpenMP trace info of an OpenMP application, pass --omp:

```
$ AMDuProfCLI collect --omp --config tbp -o /tmp/openmp_trace <path-to-openmp-exe>
```

- Launch *AMDTClassicMatMul-bin* and collect the memory accesses for false cache sharing:

```
$ AMDuProfCLI collect --config memory -o /tmp/cpuprof-mem AMDTClassicMatMul-bin
```

- Collect MPI profiling information:

```
$ mpirun -np 4 ./AMDuProfCLI collect --config assess --mpi --output-dir /tmp/cpuprof-mpi /tmp/namd <parameters>
```

- Collect the samples for PMCx076 and PMCx0C0, but collect the call graph info only for PMCx0C0:

```
$ AMDuProfCLI collect -e event=pmcx76,interval=250000 -e event=pmcxc0,interval=250000,call-graph -o /tmp/cpuprof-pmc AMDTClassicMatMul-bin
```

- Launch *AMDTClassicMatMul-bin* and collect the samples for predefined event RETIRED_INST and L1_DC_REFILLS.ALL events:

```
$ AMDuProfCLI collect -e event=RETIRED_INST,interval=250000 -e event=L1_DC_REFILLS.ALL,user=1,os=0,interval=250000 -o /tmp/cpuprof-pmc AMDTClassicMatMul-bin
```

- Launch *AMDTClassicMatMul-bin* and collect all the OS trace events:

```
$ AMDuProfCLI collect --trace os -o /tmp/cpuprof-os AMDTClassicMatMul-bin
```

- Launch *AMDTClassicMatMul-bin* and collect syscall taking more than or equal to 1ms:

```
$ AMDuProfCLI collect --trace os=syscall:1000 -o /tmp/cpuprof-os AMDTClassicMatMul-bin
```

- Launch *AMDTClassicMatMul-bin* and collect the GPU Traces for hip domain:

```
$ AMDuProfCLI collect --trace gpu=hip -o /tmp/cpuprof-gpu AMDTClassicMatMul-bin
```

- Launch *AMDTClassicMatMul-bin* and collect the GPU Traces for hip and hsa domain:

```
$ AMDuProfCLI collect --trace gpu -o /tmp/cpuprof-gpu AMDTClassicMatMul-bin
```

- Launch *AMDTClassicMatMul-bin*, collect the TBP samples and GPU Traces for hip domain:

```
$ AMDuProfCLI collect --config tbp --trace gpu=hip -o /tmp/cpuprof-gpu AMDTClassicMatMul-bin
```

- Launch *AMDTClassicMatMul-bin* and collect the GPU samples:

```
$ AMDuProfCLI collect --config gpu -o /tmp/cpuprof-gpu AMDTClassicMatMul-bin
```

- Launch *AMDTClassicMatMul-bin*, collect the GPU samples and OS Traces:

```
$ AMDuProfCLI collect --config gpu --trace os -o /tmp/cpuprof-gpu-os AMDTClassicMatMul-bin
```

- Launch *AMDTClassicMatMul-bin*, collect the TBP and GPU samples:

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

- Launch *AMDTClassicMatMul-bin* and collect the function count of malloc() called:

```
$ AMDuProfCLI collect --trace os=funccount --func c:malloc -o /tmp/cpuprof-os  
AMDTClassicMatMul-bin
```

- Launch *AMDTClassicMatMul-bin* and collect the context switches, syscalls, pthread API tracing, and function count of malloc() called:

```
$ AMDuProfCLI collect --trace os --func c:malloc -o /tmp/cpuprof-os AMDTClassicMatMul-bin
```

- Collect the system wide function count of malloc(), calloc(), and kernel functions that match the pattern 'vfs_read*':

```
$ AMDuProfCLI collect --trace os --func c:malloc,calloc,kernel:vfs_read* -o /tmp/cpuprof-os -  
a -d 10
```

6.5 Report Command

The report command generates a report in readable format by processing the raw profile data files or from the (processed) database files available in the specified directory.

Synopsis:

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

Common Usages:

```
$ AMDuProfCLI report -i <session-dir path>
```

6.5.1 Options

Table 27. AMDuProfCLI Report Command Options

Option	Description
-h --help	Displays this help information on the console/terminal.
-i --input-dir <directory-path>	Path to the directory containing collected data.
--detail	Generate detailed report.
--group-by <section>	Specify the report to be generated. The supported report options are: <ul style="list-style-type: none"> • process: Report process details • module: Report module details • thread: Report thread details This option is applicable only with --detail option. The default is group-by process.
-p --pid <PID,...>	Generate report for the specified PIDs. The process IDs are separated by comma.
-g	The print callgraph. Use with the option --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. The default value is 10.
--view <view-config>	Report only the events present in the given view file. Use the command <code>info --list view-configs</code> to get the list of supported view-configs.
--inline	Show inline functions for C, C++ executables. <p>Notes:</p> <ol style="list-style-type: none"> 1. This option is not supported on Windows. 2. 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 the source statements.
--src-path <path1;...>	Source file directories (semicolon separated paths). Multiple use of --src-path is allowed.
--disasm	Generate a detailed function report with assembly instructions.
--disasm-only	Generate the function report with only assembly instructions.

Table 27. AMDuProfCLI Report Command Options

Option	Description
-s --sort-by <EVENT>	<p>Specify the Timer, PMC, or IBS event on which the reported profile data will be sorted with arguments in the form of comma separated key=value pairs. The supported keys are:</p> <ul style="list-style-type: none"> • event=<timer ibs-fetch ibs-op pmcxNNN>, where NNN is hexadecimal Core PMC event id. • umask=<unit-mask> • cmask=<count-mask> • inv=<0 1> • user=<0 1> • os=<0 1> <p>Use the command <code>info --list pmu-events</code> for the list of supported PMC events.</p> <p>Details about the arguments:</p> <ul style="list-style-type: none"> • umask — Unit mask in decimal or hexadecimal, applicable only to the PMC events. • cmask — Count mask in decimal or hexadecimal, applicable only to the PMC events. • user, os — User and OS mode. Applicable only to the PMC events. • inv — Invert Count Mask, applicable only to the PMC events <p>Multiple occurrences of <code>--sort-by (-s)</code> are not allowed.</p>
--agg-interval <INTERVAL>	Accumulation of samples for the given INTERVAL in milliseconds. Default INTERVAL is 1 second. If the interval value set to 0, granularity of the data aggregation would be the complete profile duration.
--time-filter <T1:T2>	Restricts report generation to the time interval between T1 and T2. Where, T1 and T2 are time in seconds from profile start time.
--agg-interval <INTERVAL>	Accumulation of samples for the given INTERVAL in milliseconds. Default INTERVAL is 1 millisecond.
--imix	Generate instruction MIX report. It is only supported for IBS config and IBS events profiling. It is only supported for the native binaries.
--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 the number of events occurred.
--show-all-cachelines	Show all the cachelines in the report sections for cache analysis. By default, only the cachelines accessed by more than one process/thread are listed. Supported only for memory config report on Windows and Linux platforms.
--bin-path <path>	Binary file path, multiple usage of <code>--bin-path</code> is allowed.
--src-path <path>	Source file path, multiple usage of <code>--src-path</code> is allowed.
--symbol-path <path1;...>	Debug Symbol paths (semicolon separated). Multiple use of <code>--symbol-path</code> is allowed.

Table 27. AMDuProfCLI Report Command Options

Option	Description
--report-output <path>	Write a report to a file. If the path has a .csv extension, it is assumed to be a file path and used as it is. If the .csv extension is not used, then the path is assumed to be a directory and the report file is generated in the directory with the default name.
--stdout	Print the report to a console or terminal.
--retranslate	Perform the re-translation of collected data files with a different set of translation options.
--log-path <path-to-log-dir>	Specify the path where the log file should be created. If this option is not provided, the log file will be created either in the path set by AMDUPROF_LOGDIR environment variable or \$TEMP path (Linux, FreeBSD) or %TEMP% path (on Windows) by default. The log file name will be of the format \$USER-AMDuProfCLI.log (on Linux, FreeBSD) or %USERNAME%-AMDuProfCLI.log (on Windows).
--enable-log	Enable additional logging with log file.
--enable-logts	Capture the timestamp of the log records. This option should be used with --enable-log option.

6.5.2 Windows Specific Options

Table 28. AMDuProfCLI Report Command - Windows Specific Options

Option	Description
--symbol-server <path1;...>	Symbol Server directories (semicolon separated paths). For example, Microsoft Symbol Server (https://msdl.microsoft.com/download/symbols). Multiple use of --symbol-server is allowed.
--symbol-cache-dir <path>	The path to store the symbol files downloaded from the Symbol Servers.

6.5.3 Linux Specific Options

Table 29. AMDuProfCLI Report Command - Linux Specific Options

Option	Description
--host <hostname>	<p>This option is used along with the --input-dir option. Generates report belonging to a specific host. The supported options are:</p> <ul style="list-style-type: none"> • <hostname>: Report process belonging to a specific host. • all: Report all the processes. <p><i>Note: If --host is not used, only the processes belonging to the system from which report is generated is reported. In case, the system is a master node in a cluster, the report will be generated for the lexicographically first host in that cluster.</i></p>
--category <PROFILE>	<p>Generate report only for specific profiling category. Comma separated multiple categories can be specified. If this option is not used, then report for all categories gets generated. Multiple instance of --category is allowed. Supported categories are:</p> <ul style="list-style-type: none"> • cpu – Generate report specific to CPU Profiling. • mpi – Generate report specific to MPI Tracing. • openmp – Generate report specific to OpenMP Tracing. • os – Generate report specific to OS Tracing. • gputrace – Generate report specific to GPU Tracing. • gpuprof – Generate report specific to GPU Profiling. <p>Example:</p> <pre>--category cpu,mpi,os,gputrace,gpuprof --category mpi --category cpu --category os --category gputrace --category gpuprof</pre>
--funccount-interval <funccount-interval>	<p>Specify the time interval in seconds to list the function count detail report. If this option is not specified, the function count will be generated for the entire profile duration.</p>

6.5.4 Examples

Windows

- Generate report from the raw datafile:

```
C:\> AMDuProfCLI.exe report -i c:\Temp\cpuprof-tbp\<SESSION-DIR>
```

- Generate IMIX report from the raw datafile:

```
C:\> AMDuProfCLI.exe report --imix -i c:\Temp\cpuprof-imix\<SESSION-DIR>
```

- Generate report from the raw datafile sorted on pmc event:

```
C:\> AMDuProfCLI.exe report -s event=pmcxc0,user=1,os=0 -i c:\Temp\cpuprof-ebp\<SESSION-DIR>
```

- Generate report from the raw datafile sorted on ibs-op event:

```
C:\> AMDuProfCLI.exe report -s event=ibs-op -i c:\Temp\cpuprof-ibs\<SESSION-DIR>
```

- Generate report from the raw datafile for power samples:

```
C:\> AMDuProfCLI.exe report -i c:\Temp\pwrprof-swp\<SESSION-DIR>
```

- Generate report with Symbol Server paths:

```
C:\> AMDuProfCLI.exe report --symbol-path C:\AppSymbols;C:\DriverSymbols --symbol-server
http://msdl.microsoft.com/download/symbols --symbol-cache-dir C:\symbols -i c:\Temp\cpuprof-
tbp\<SESSION-DIR>
```

- Generate report from the raw datafile on one of the predefined views:

```
C:\> AMDuProfCLI.exe report --view ipc_assess -i c:\Temp\pwrprof-swp\<SESSION-DIR>
```

- Generate report from the raw datafile providing the source and binary paths:

```
C:\> AMDuProfCLI.exe report --bin-path Examples\AMDTClassicMatMul\bin\ --src-path
Examples\AMDTClassicMatMul\ -i c:\Temp\cpuprof-tbp\<SESSION-DIR>
```

Linux

- Generate report from the raw datafile:

```
$ AMDuProfCLI report -i /tmp/cpuprof-tbp/<SESSION-DIR>
```

- Generate IMIX report from the raw datafile:

```
$ AMDuProfCLI report --imix -i /tmp/cpuprof-imix/<SESSION-DIR>
```

- Generate report from the raw datafile sorted on pmc event:

```
$ AMDuProfCLI report -s event=pmcx0,user=1,os=0 -i /tmp/cpuprof-ebp/<SESSION-DIR>
```

- Generate report from the raw datafile sorted on ibs-op event:

```
$ AMDuProfCLI report -s event=ibs-op -i /tmp/cpuprof-ibs/<SESSION-DIR>
```

- Generate OS Trace report from the raw datafile:

```
$ AMDuProfCLI report -i /tmp/cpuprof-os/<SESSION-DIR> --category os
```

- Generate GPU Trace report from the raw datafile:

```
$ AMDuProfCLI report -i /tmp/cpuprof-gpu/<SESSION-DIR> --category gputrace
```

- Generate GPU Profile report from the raw datafile:

```
$ AMDuProfCLI report -i /tmp/cpuprof-gpu/<SESSION-DIR> --category gpuprof
```

6.6 Translate Command

The `translate` command processes the raw profile data and generates the samples info database files. These databases can be imported to GUI or CLI and used for generating the report.

Synopsis:

```
AMDuProfCLI translate [<options>]
```

Common Usages:

```
$ AMDuProfCLI translate -i <session-dir path>
```

6.6.1 Options

Following table lists the AMDuProfCLI `translate` command options:

Table 30. AMDuProfCLI Translate Command Options

Option	Description
<code>-h --help</code>	Displays the help information.
<code>-i --input-dir <directory-path></code>	Path to the directory containing collected data.
<code>--time-filter <T1:T2></code>	Restricts the processing to the time interval between T1 and T2, where T1, T2 are time in seconds from profile start time.
<code>--agg-interval <INTERVAL></code>	Accumulation of samples for the given INTERVAL in milliseconds. The default INTERVAL is 1 second. If the interval value set to 0, granularity of the data aggregation would be the complete profile duration.
<code>--bin-path <path></code>	Binary file path. Multiple use of <code>--bin-path</code> is allowed.
<code>--symbol-path <path></code>	Debug symbol path. Multiple instances of <code>--symbol-path</code> are allowed.
<code>--inline</code>	Inline function extraction for C and C++ executables. <i>Notes:</i> 1. This option is not supported on Windows. 2. Using this option will increase the time taken to generate the report.
<code>--retranslate</code>	Re-translate the collected data files with a different set of translation options.
<code>--log-path <path-to-log-dir></code>	Specify the path where the log file should be created. If this option is not provided, the log file will be created either in the path set by <code>AMDUPROF_LOGDIR</code> environment variable or <code>%TEMP%</code> path by default. The log file name will be of the format <code>\$USER-AMDuProfCLI.log</code> (on Linux, FreeBSD) or <code>%USERNAME%-AMDuProfCLI.log</code> (on Windows).
<code>--enable-log</code>	Enable additional logging with log file.
<code>--enable-logts</code>	Capture the timestamp of the log records. This option should be used with the <code>--enable-log</code> option.

6.6.2 Windows Specific Options

Following table lists the Windows specific options of the `translate` command:

Table 31. Translate Command - Windows Specific Options

Option	Description
<code>--symbol-server <path1;...></code>	Links to Symbol Server, for example, Microsoft Symbol Server (https://msdl.microsoft.com/download/symbols). Multiple instances of <code>--symbol-server</code> are allowed.
<code>--symbol-cache-dir <path></code>	Path to save the symbols downloaded from the Symbol Servers.

6.6.3 Linux Specific Options

Following table lists the Linux specific options of the `translate` command:

Table 32. Translate Command - Linux Specific Options

Option	Description
<code>--category <PROFILE></code>	Process only a specific profiling category. Comma separated multiple categories can be specified. If this option not used, then all categories raw data files are processed. Multiple instances of <code>--category</code> are allowed. The supported categories are: <ul style="list-style-type: none"> • <code>cpu</code> - CPU Profiling • <code>mpi</code> - MPI Tracing • <code>openmp</code> – Generate report specific to OpenMP Tracing. • <code>os</code> - OS Tracing • <code>gputrace</code> - GPU Tracing • <code>gpuprof</code> - GPU Profiling Example: <pre>--category cpu,mpi,os,gputrace,gpuprof --category mpi --category cpu --category os --category gputrace --category gpuprof</pre>
<code>--host <hostname></code>	This option is used with the <code>--input-dir</code> option. It processes samples belonging to a specific host. The supported options are: <p><code><hostname></code>: Translate only the processes belonging to a specific host.</p> <p><code>all</code>: Translate all processes</p> <p>Note: <i>If <code>--host</code> is not used, then only the processes belonging to the current system is translated. In case the system is a master node in a cluster, then processing will be done for the lexicographically first host in that cluster.</i></p>
<code>--kallsyms-path <path></code>	Path to the file containing kallsyms info. If no path is provided, it defaults to <code>/proc/kallsyms</code> .
<code>--vmlinux-path <path></code>	Path to the Linux kernel debug info file. If no path provided, it searches for the debug info file in the default download path.

6.6.4 Examples

Windows

- Process all the raw data files:

```
> AMDuProfCLI.exe translate -i c:\Temp\cpuprof-tbp\<SESSION-DIR>
```

- Process the raw data files with Symbol Server paths:

```
> AMDuProfCLI.exe translate --symbol-path C:\AppSymbols;C:\DriverSymbols --symbol-server
http://msdl.microsoft.com/download/symbols --symbol-cache-dir C:\symbols -i c:\Temp\cpuprof-
tbp\<SESSION-DIR>
```

- Process the raw data files with the source and binary path:

```
> AMDuProfCLI.exe translate --bin-path Examples\AMDTClassicMatMul\bin\ --src-path
Examples\AMDTClassicMatMul\ -i c:\Temp\cpuprof-tbp\<SESSION-DIR>
```

Linux

- Process all the raw data files:

```
$ AMDuProfCLI translate -i /tmp/cpuprof-tbp/<SESSION-DIR>
```

- Process the OS Trace raw data file:

```
$ AMDuProfCLI translate -i /tmp/cpuprof-os/<SESSION-DIR> --category os
```

- Process the GPU Trace raw data file:

```
$ AMDuProfCLI translate -i /tmp/cpuprof-gpu/<SESSION-DIR> --category gputrace
```

6.7 Timechart Command

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

Note: The timechart command is supported only on Windows and Linux.

Synopsis:

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

<PROGRAM> — Denotes the application to be launched before starting the power metrics collection.

<ARGS> — Denotes the list of arguments for the launch application.

Common Usages:

```
$ AMDuProfCLI timechart --list
```

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

6.7.1 Options

Table 33. AMDuProfCLI Timechart Command Options

Option	Description
-h --help	Displays this help information.
--list	Displays all the supported devices and categories.
-e --event <type...>	Collect counters for specified combination of device type and/or category type. Use command <code>timechart --list</code> for the list of supported devices and categories. <i>Note: Multiple occurrences of -e is allowed.</i>
-t --interval <n>	Sampling interval n in milliseconds. The minimum value is 10ms.
-d --duration <n>	Profile duration n in seconds.
--affinity <core...>	The core affinity. Comma separated list of core-ids. Ranges of core-ids is also be specified, for example, 0-3. The default affinity is all the available cores. The 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-dir <dir>	Output directory path.

6.7.2 Examples

Windows

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

```
C:\> AMDuProfCLI.exe timechart --event power --interval 100 --duration 10
```

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

```
C:\> AMDuProfCLI.exe timechart --event frequency -o C:\Temp\output --interval 500 --duration 10
```

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

```
C:\> AMDuProfCLI.exe timechart --event core=0-3,frequency -o C:\Temp\PowerOutput --interval 500 -duration 10 --format txt
```

Linux

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

```
$ ./AMDuProfCLI timechart --event power --interval 100 --duration 10
```

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

```
$ ./AMDuProfCLI timechart --event frequency -o /tmp/PowerOutput --interval 500 --duration 10
```

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

```
$ ./AMDuProfCLI timechart --event core=0-3,frequency -o /tmp/PowerOutput --interval 500 --duration 10 --format txt
```

6.8 Info Command

This command fetches the generic information about the system, PMC event details, predefined event details, and so on.

Synopsis:

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

Common Usages:

```
$ AMDuProfCLI info --system
```

6.8.1 Options

Table 34. AMDuProfCLI Info Command Options

Option	Description
-h --help	Displays the help information.
--list <type>	Lists the supported items for the following types: <ul style="list-style-type: none"> collect-configs: Predefined profile configurations that can be used with collect --config option. predefined-events: List of the supported predefined events that can be used with collect --event option. pmu-events: Raw PMC events that can be used with collect --event option. Alternatively, info --pmu-event all can be used to print information of all the supported events. cacheline-events: List of event aliases to be used with report --sort-by option for cache analysis. It is supported only on Windows and Linux platforms. view-configs: List the supported data view configurations that can be used with report --view option.
--collect-config <name>	Displays the details of the given profile configuration used with collect --config <name> option. Use info --list collect-configs command for the details on the supported profile configurations.
--view-config <name>	Displays the details of the given view configuration used in the report generation option report --view <name>. Use info --list view-configs command for the details on the supported data view configurations.
--pmu-event <event>	Displays the details of the given pmu event. Use command info --list pmu-events for the list of supported PMC events.
--system	Displays the processor information of this system.

Following table lists the Linux specific info command options:

Table 35. AMDuProfCLI Info Command - Linux Specific Options

Option	Description
--list <type>	Lists the supported items for the following types: <ul style="list-style-type: none"> ostrace-events: List of OS Trace events that can be used with collect --trace os option. gpu-events: List of GPU events can be used in gpu profile configuration.
--bpf	Displays details of the BPF support and BCC Installation.

6.8.2 Examples

Use the following commands to:

- Print the system details:

```
C:\> AMDuProfCLI.exe info --system
```

- Print the list of predefined profiles:

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

- Print the list of PMU events:

```
C:\> AMDuProfCLI.exe info --list pmu-events
```

- Print the list of predefined report views:

```
C:\> AMDuProfCLI.exe info --list view-configs
```

- Print details of predefined profile such as “`assess_ext`”:

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

- Print the details of the pmu-event such as `PMCx076`:

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

- Print details of view configuration such as `ibs_op_overall`:

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

Chapter 7 Performance Analysis

7.1 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 one 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
 - Thread concurrency (Windows only, requires admin privilege)
 - Thread names (Windows and Linux only)
- Profile Scope
 - Per-Process — launch an application and profile that process and 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 (5.0, 6.0, and Framework), FORTRAN, and Assembly applications
- Various software components — applications, dynamically linked/loaded modules, driver, and OS kernel modules
- Profile data is attributed at various granularities
 - Process, Thread, Load Module, Function, Source line, or Disassembly
 - To correlate the profile data to Function and Source line, debug information emitted by the compiler is required
 - C++ and Java in-lined functions
- Processed profile data is stored in a database that can be used for generating a report 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 UI elements to analyze and view the profile data at various granularities:

- Hot spots summary
- Session Information
- Thread concurrency graph (Windows only and requires admin privileges)
- Process and function analysis
- Source and disassembly analysis
- Bottom-up Call Path — bottom-up function callchain view in the Function Hotspots view
- 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 and MPI profile data
- Timeline Visualizer — timeline views for MPI API trace, OS event trace, and GPU trace information
- 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.

7.2 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.

7.2.1 Configuring and Starting Profile

Complete the following steps to configure and start a profile:

1. Click **PROFILE > Start Profiling** to navigate to the **Select Profile Target** screen.
2. Select the required profile target, click the **Next** button.

The **Select Profile Type** screen is displayed.

- From the **Select Profile Type** drop-down, select the **CPU Profile** from the drop-down.

The following screen is displayed:

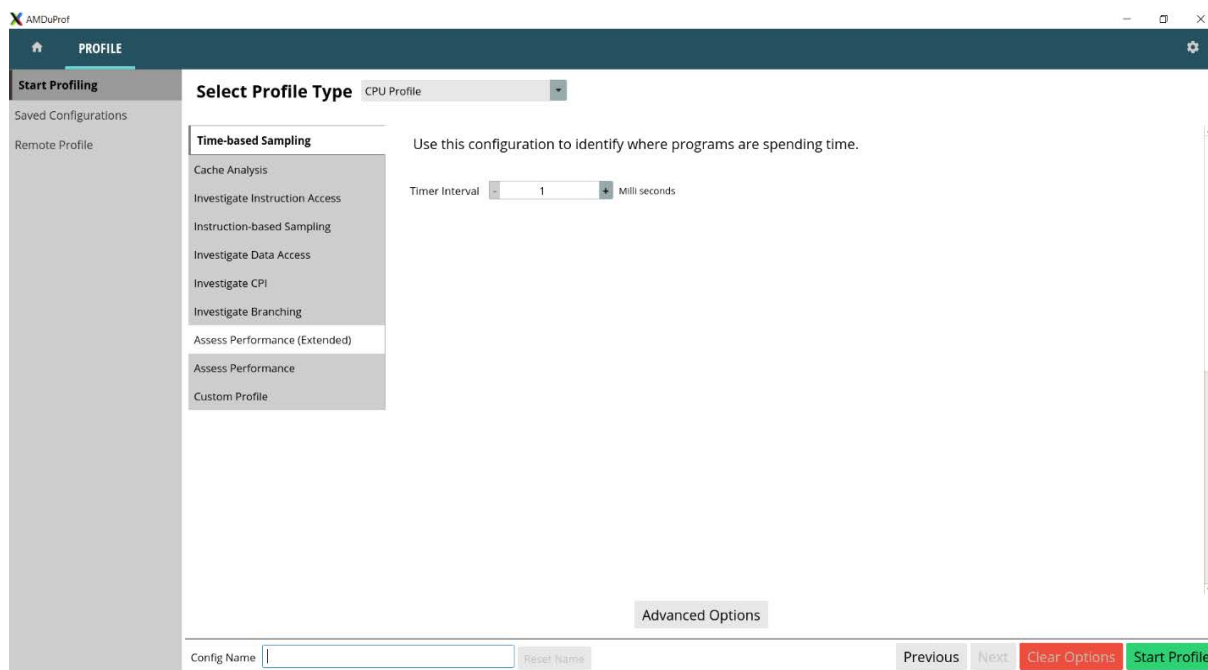


Figure 29. Time-based Profile – Configure

- Select **Time-based Sampling** in the left vertical pane.
- Click **Advanced Options** to enable call-stack, set symbol paths (if the debug files are in different locations) and other options. Refer the section “Advanced Options” on page 67 section for more information on this screen.
- 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 the profile data collection screen is displayed.

7.2.2 Analyzing Profile Data

Complete the following steps to analyze the profile data:

- When the profiling stops, the collected raw profile data will be processed automatically and the **Hot Spots** screen of the **Summary** page is displayed. The hotspots are shown for the **Timer** samples. Refer the section “Overview of Performance Hotspots” on page 70 for more information on this screen.
- Click **ANALYZE** on the top horizontal navigation bar to go to the **Function HotSpots** screen. Refer the section “Function HotSpots” on page 71 for more information on this screen.

3. Click **ANALYZE > Metrics** to display the profile data table at various granularities - Process, Load Modules, Threads, and Functions. Refer the section “Process and Functions” on page 73 for more information on this screen.
4. Double-click any entry on the **Functions** table in **Metrics** screen to load the source tab for that function in **SOURCES** page. Refer the section “Source and Assembly” on page 75 for more information on this screen.

7.3 Analysis with Event-based Profiling

In this profile, AMD 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.

7.3.1 Configuring and Starting Profile

Complete the following steps to configure and start a profile:

1. Click **PROFILE > Start Profiling** to navigate to the **Select Profile Target** screen.
2. Select the required profile target, click the **Next** button.

The **Start Profiling** screen is displayed as follows:

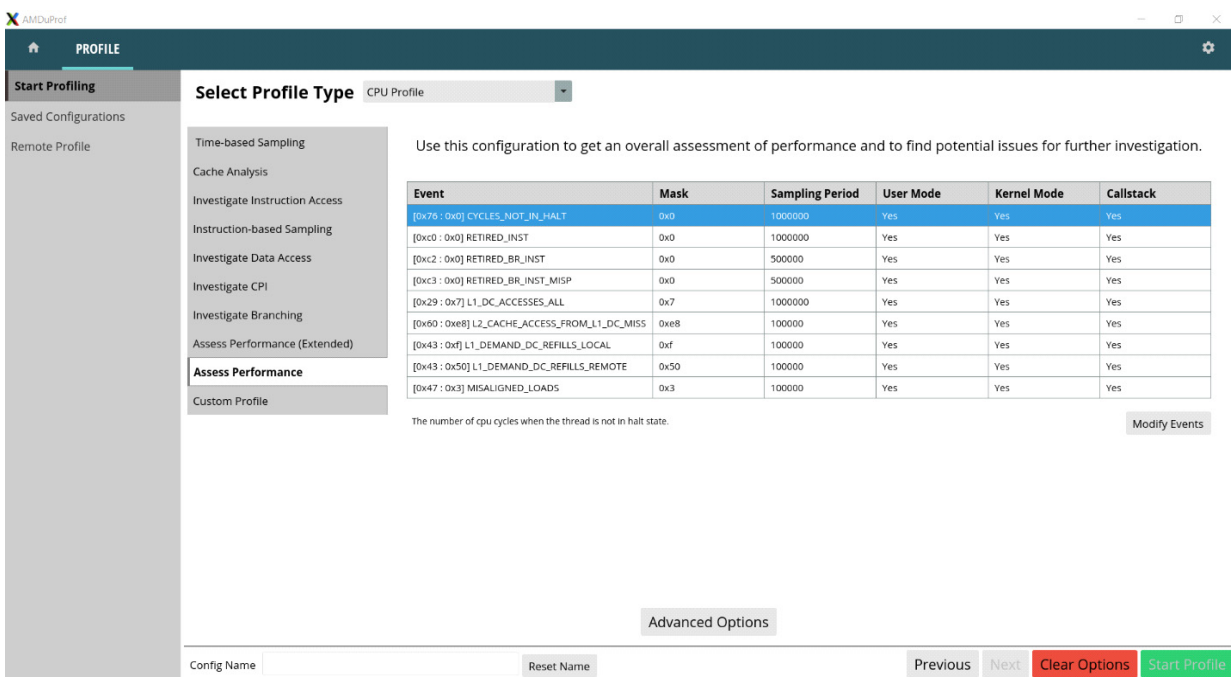


Figure 30. Event-based Profile – Configure

3. From the **Select Profile Type** drop-down, select the **CPU Profile**.

4. Select **Assess Performance** in the left vertical pane. Refer the section “Predefined Sampling Configuration” on page 28 for EBP based predefined sampling configurations.
5. Click **Advanced Options** to enable call-stack, set symbol paths (if the debug files are in different locations) and other options. Refer the section “Advanced Options” on page 67 for more information on this screen.
6. Once all the options are set, the **Start Profile** button at the bottom will be enabled. Click it to start the profile.

After the profile initialization the profile data collection screen is displayed.

7.3.2 Analyzing Profile Data

Complete the following steps to analyze the profile data:

1. When the profiling stops, the collected raw profile data will be processed automatically and the **Hot Spots** screen of the **Summary** page is displayed. Refer the section “Overview of Performance Hotspots” on page 70 for more information on this screen.
2. Click **ANALYZE** on the top horizontal navigation bar to go to the **Function HotSpots** screen. Refer the section “Function HotSpots” on page 71 for more information on this screen.
3. Click **ANALYZE > Metrics** to display the profile data table at various granularities - Process, Load Modules, Threads, and Functions. Refer the section “Process and Functions” on page 73 for more information on this screen.
4. Double-click any entry on the **Functions** table in **Metrics** screen to load the source tab for that function in **SOURCES** page. Refer the section “Source and Assembly” on page 75 for more information on this screen.

7.4 Analysis with Instruction-based Sampling

In this profile, AMD uProf uses the IBS supported by the AMD x64-based processor to diagnose the performance issues in hot spots. It collects data on how instructions behave on the processor and in the memory sub-system.

7.4.1 Configuring and Starting Profile

Complete the following steps to configure and start a profile:

1. Click **PROFILE > Start Profiling** to navigate to the **Select Profile Target** screen.
2. Select the required profile target, click the **Next** button.
3. From the **Select Profile Type** drop-down, select the **CPU Profile**.
4. Select **Instruction-based Sampling** in the left vertical pane. Refer the section “Predefined Sampling Configuration” on page 28 for IBS based predefined sampling configurations.

5. Click **Advanced Options** to enable call-stack, set symbol paths (if the debug files are in different locations) and other options. Refer the section “Advanced Options” on page 67 for more information on this screen.
6. Once all the options are set, the **Start Profile** button at the bottom will be enabled. Click it to start the profile.

After the profile initialization the profile data collection screen is displayed.

7.4.2 Analyzing Profile Data

Complete the following steps to analyze the profile data:

1. When the profiling stops, the collected raw profile data will be processed automatically and you the **Hot Spots** screen of the **Summary** page is displayed. Refer the section “Overview of Performance Hotspots” on page 70 for more information on this screen.
2. Click **ANALYZE** on the top horizontal navigation bar to go to the **Function HotSpots** screen. Refer the section “Process and Functions” on page 73 for more information on this screen.
3. Click **ANALYZE > Metrics** to display the profile data table at various granularities - Process, Load Modules, Threads, and Functions. Refer the section “Process and Functions” on page 73 for more information on this screen.
4. Double-click any entry on the **Functions** table in **Metrics** screen to load the source tab for that function in **SOURCES** page. Refer the section “Source and Assembly” on page 75 for more information on this screen.

7.5 Analysis with Call Stack Samples

The call stack samples 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.

Note: *Java call stack profiling is supported only on Linux platforms.*

To enable call stack sampling, complete the following steps:

1. Select profile target and profile type.

2. Click on **Advanced Options** button to turn on the **Enable CSS** option in **Call Stack Options** pane as follows:

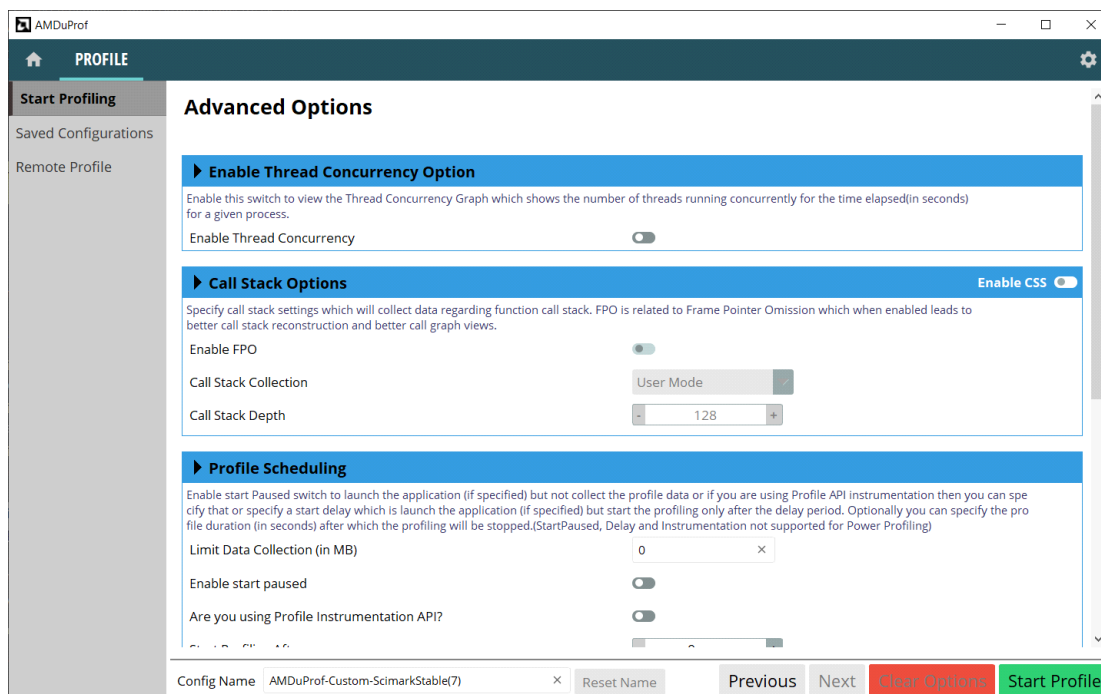


Figure 31. Start Profiling - Advanced Options

Refer the section “Advanced Options” on page 67 for more information on this screen.

Note: If the application is compiled with higher optimization levels and frame pointers are not displayed, **Enable FPO** option can be turned on. On Linux, this will increase the size of the raw profile file size.

7.5.1 Flame Graph

Flame Graph provides a stack visualizer based on call stack samples. The **Flame Graph** is available in the **ANALYZE** page to analyze the call stack samples to identify hot call-paths. To access it, navigate to **ANALYZE > Flame Graph** in the left vertical pane.

Refer the section “Flame Graph” on page 76 for more information on this screen.

The following figure shows a sample flame graph:

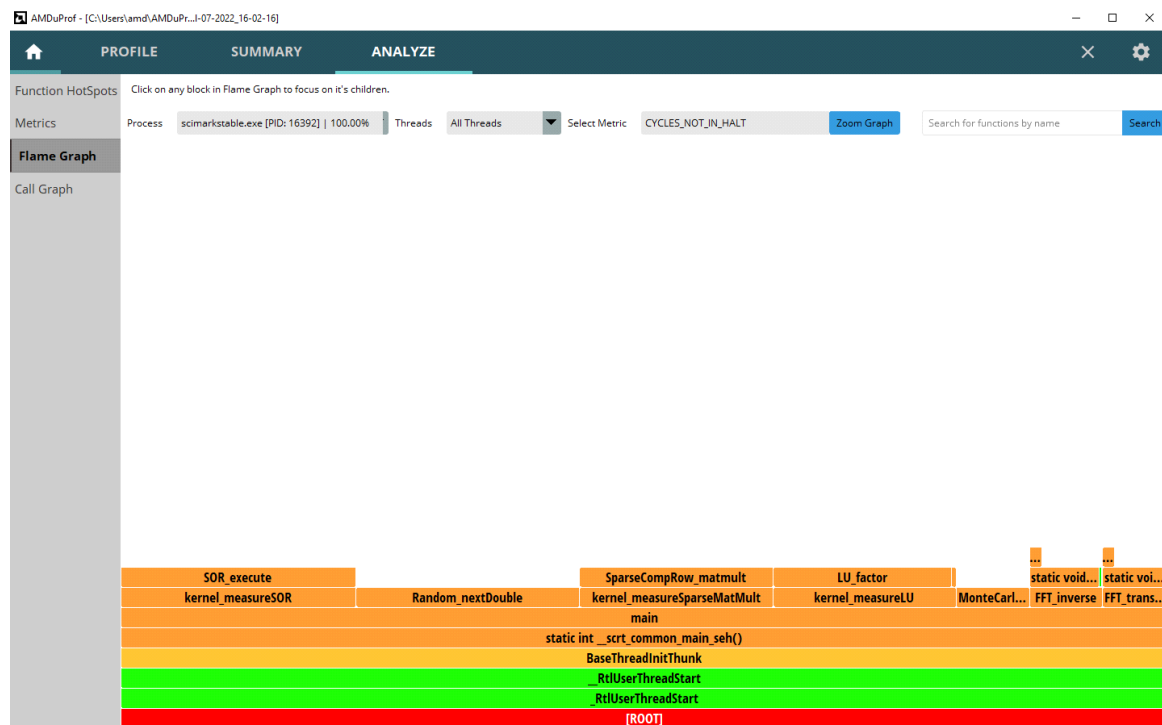


Figure 32. ANALYZE - Flame Graph

The flame graph can be displayed based on the **Process** and **Select Metric** drop-downs. Also, it has the function search box to search and highlight the given function name.

7.5.2 Call Graph

Call Graph provides a butterfly view of call graph based on call-stack samples. The **Call Graph** screen will be available in **ANALYZE** page to analyze the call-stack samples to identify hot call-paths. To access it, click **ANALYZE > Call Graph** in the left vertical pane.

Refer the section “Call Graph” on page 78 for more information on this screen.

The following figure shows a sample call graph:

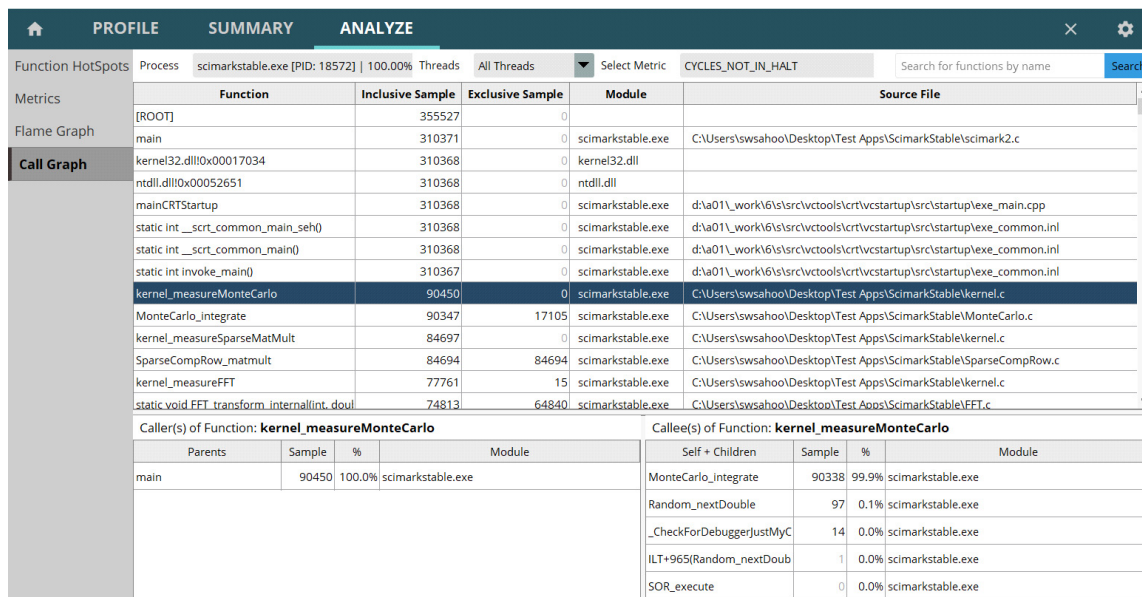


Figure 33. ANALYZE - Call Graph

You can browse the data based on **Process** and **Select Metric** drop-downs. The top central table displays call-stack samples for each function. Click on any function to update the bottom two **Caller(s)** and **Callee(s)** tables. These tables display the callers and callees respectively of the selected function.

7.6 Profiling a Java Application

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

AMD uProf provides JVMTI Agent libraries: *AMDJvmtiAgent.dll* on Windows and *libAMDJvmtiAgent.so* on Linux. This JvmtiAgent library must be loaded during start up of the target JVM process.

7.6.1 Launching a Java Application

If the Java application is launched by AMD uProf, the tool would pass the AMDJvmtiAgent library to JVM using Java -agentpath option. AMD uProf would be able to collect the profile data and attribute the samples to interpreted Java functions.

To profile a Java application, use the following sample command:

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

To generate report, pass the following source file path:

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

7.6.2 Attaching a Java Process to Profile

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

To profile an already running Java process, pass `-agentpath <path to agent lib>` option while launching Java application. So that, AMD uProf can attach to the Java PID to collect profile data later on.

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 AMD uProf GUI or AMD uProf CLI to attach to this process and profile.

7.6.3 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 the section “Source and Assembly” on page 75 for more information on source screen.

The following figure shows the source view of the Java method:

Home

PROFILE

SUMMARY

ANALYZE

TOOLS

Close

Settings

jnt.scimark2.SparseCompRow::matmult(double[],double[],int[],int[],double[],int)

Select View

Overall assessment

Value Type

Sample Count

Process

java.exe (PID 25196) | 100.00%

Threads

Thread-44088 | 100.00%

Show Assembly

Line	Source	RETIRED_INST	CYCLES_NOT_IN_HALT	HE ACCESS FROM L1	MISALIGNED LOAD	RETIRED_BR_INST_MIS	IPC
40	sum += x[col[i]] * val[i];						
41	y[r] = sum;	251	168	47		1	1.49
...Non-contiguous source line(s)...							
38	int rowRp1 = row[r+1];	5526	2014	960		3	2.74
...Non-contiguous source line(s)...							
37	int rowR = row[r];	319	257	64		1	1.24
38	int rowRp1 = row[r+1];						
39	for (int i=rowR; i<rowRp1; i++)	6857	2472	1181		4	2.77
40	sum += x[col[i]] * val[i];	58421	20649	10041		27	2.83
41	y[r] = sum;						
42	}						
43	}						
44	}						

< >

Address	Line	Assembly	RETIRED_INST	CYCLES_NOT_IN_HALT	HE ACCESS FROM L1	MISALIGNED LOAD	RETIRED_BR_INST_MIS	IPC
0x13f	40	vaddsd xmm0,xmm0,xmm1	5606	2117	972		4	2.65
0x143	40	inc r10d	7073	2517	1239		3	2.81
0x146	39	cmp r10d,ebp	14	10	2			1.40
0x149	39	j1 0000000000000124h	17	8	6			2.13
0x14b	39	mov ebp,ecx						
0x14d	39	add ebp,fdh	83	30	12			2.77
0x150	39	cmp ecx,ebp	15	13	3			1.15
0x152	39	cmovl ebp,r14d	928	284	165			3.27
0x156	39	cmp r10d,ebp	28	37	1			0.76
0x159	39	jnl 00000000000001e5h	2714	1042	496		2	2.60
0x15f	39	nop						
0x160	40	mov eax,[rdi+r10*4+10h]	43	38	4			1.13
0x165	40	cmp eax,r9d	48	52	9			0.92

< >

Figure 34. Java Method - Source View

7.6.4 Java Call Stack and Flame Graph

Note: Java call stack profiling is supported only on Linux platforms.

To collect call stack for profiling Java application, use the following command:

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

The following figure shows the flame graph of a Java application:

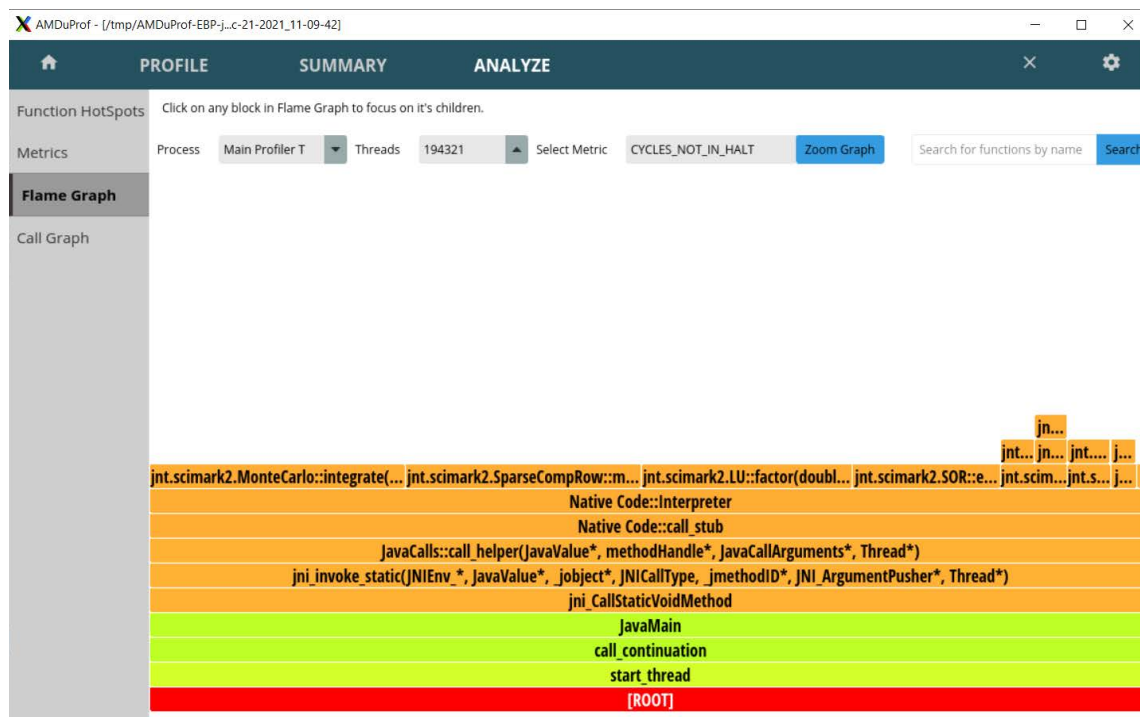


Figure 35. Java Application - Flame Graph

7.7 Cache Analysis

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

At a high-level, this feature will report:

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

7.7.1 Supported Metrics

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

Table 36. IBS OP Derived Metrics

Metric	Description
IBS_LOAD_STORE	Total Loads and stores sampled
IBS_LOAD	Total Loads
IBS_STORE	Total Stores
IBS_DC_MISS_LAT	Accumulated load latencies for the loads to cache lines
IBS_LOAD_DC_L2_HIT	Load operations hit in data cache or L2 cache
IBS_NB_LOCAL_CACHE_MODIFIED	Loads that were serviced from the local cache (L3) and the cache hit state was Modified
IBS_NB_LOCAL_CACHE_OWNED	Loads that were serviced from the local cache (L3) and the cache hit state was Owned
IBS_NB_LOCAL_CACHE_MISS	Loads that were missed in local cache (L3) and serviced by remote cache, local, or remote DRAM
IBS_NB_REMOTE_CACHE_MODIFIED	Loads that were serviced from the remote cache (L3) and the cache hit state was Modified
IBS_NB_REMOTE_CACHE_OWNED	Loads that were serviced from the remote cache (L3) and the cache hit state was Owned
IBS_NB_LOCAL_DRAM	Loads that hit in local memory (Memory channels attached to local socket or local CCD)
IBS_NB_REMOTE_DRAM	Loads that hit in remote memory (Memory channels attached to remote socket or other CCDs in the local socket)
IBS_STORE_DC_MISS	Store operations missed in data cache

7.7.2 Cache Analysis Using GUI

Configuring and Starting Profile

To perform cache analysis, complete the following steps:

1. Selecting profile target.
2. Select **Cache Analysis** profile type in **Select Profile Type** page.
3. 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 the cache-lines and it offsets with the associated metric values:

Cache Line Address/Offset/Thread/Function	IBS_NB_CACHE_MODIFIED	IBS_DC_MISS_LAT	IBS_LOAD_STORE	IBS_NB_RE	IBS_NB_LOC	IBS_LOAD	IBS_STORE	IBS_NB_LOCAL
0x3d161f60c0	279	32176	41549	0	0	6720	34829	
Offset 0x8	279	32176	6720	0	0	6720	0	
(Process: false_sharing.e) [Thread: reader_thread [TID:52975]]	279	32176	6720	0	0	6720	0	
read_func():187	279	32176	6720	0	0	6720	0	
Offset 0x10	0	0	34829	0	0	0	34829	
(Process: false_sharing.e) [Thread: writer_thread [TID:52976]]	0	0	34829	0	0	0	34829	
write_func():263	0	0	34829	0	0	0	34829	
0xa32878f0c0	12	1706	1715	0	0	148	1567	
Offset 0x8	12	1706	148	0	0	148	0	
(Process: false_sharing.e) [Thread: reader_thread [TID:52975]]	12	1706	148	0	0	148	0	
read_func():187	12	1706	148	0	0	148	0	
Offset 0x10	0	0	1567	0	0	0	1567	
(Process: false_sharing.e) [Thread: writer_thread [TID:52976]]	0	0	1567	0	0	0	1567	
write_func():263	0	0	1567	0	0	0	1567	

Figure 36. Cache Analysis

The **Cache Analysis** screen has the following options:

- **Group By** drop-down decides how the cache-line samples are grouped in the detailed table. It has the option **Cache Line Offset**.
- **ValueType** drop-down allows you to show the value in sample count.

7.7.3 Cache Analysis Using CLI

The CLI has a config type called “memory” to cache the 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 raw profile data file is created in `/tmp/cache_analysis/AMDuProf-IBS_<timestamp>/` directory.

Report Generation and Analysis

Use the following CLI command to generate the cache analysis report:

```
$ AMDuProfCLI report -i /tmp/cache_analysis/AMDuProf-IBS_<timestamp>/
```

This will generate a CSV report in `/tmp/cache_analysis/AMDuProf-IBS_<timestamp>/report.csv` and it 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 following:
 - The cache lines having a potential false sharing
 - Offsets where those accesses occur, readers and writers to those offsets
 - PID, TID, Function Name, Source File, and Line Number for those reader and writers
 - Load latency for the loads to those cache lines
 - Supported metrics

Following figure shows the **Cache Analysis** summary sections:

SHARED DATA CACHELINE SUMMARY															
IBS_LOAD_STORE:	143681														
IBS_LOAD:	89683														
IBS_STORE:	108142														
IBS_DC_MISS_LAT:	40008														
IBS_LOAD_DC_L2_HIT:	89329														
IBS_STORE_DC_MISS:	38371														
IBS_NB_LOCAL_DRAM:	0														
IBS_NB_REMOTE_DRAM:	0														
IBS_NB_CACHE_MODIFIED:	354														
IBS_NB_LOCAL_CACHE_MODIFIED:	354														
IBS_NB_REMOTE_CACHE_MODIFIED:	0														
IBS_NB_LOCAL_CACHE_OWNED:	0														
IBS_NB_REMOTE_CACHE_OWNED:	0														
IBS_NB_LOCAL_CACHE_MISS:	0														
SHARED DATA CACHELINE REPORT															
CacheLine Address	IBS_LOAD_STORE	IBS_LOAD	IBS_STORE	IBS_DC_MISS_LAT	IBS_LOAD_DC_L2_HIT	IBS_STORE_DC_MISS	IBS_NB	IBS_NF	IBS_NE	IBS_NB	IBS_NB	IBS_NB	IBS_NB	IBS_NB	IBS_NB
0x3ecd4220c0	43639	6882	36757	38615	6540	36655	0	0	342	342	0	0	0	0	0
0xb617fa40c0	1902	182	1720	1393	170	1716	0	0	12	12	0	0	0	0	0
0xb617fa4000	7	7	0	0	7	0	0	0	0	0	0	0	0	0	0

Figure 37. Cache Analysis - Summary Sections

Following figure shows the **Cache Analysis** detailed report:

SHARED DATA CACHELINE DETAILED REPORT															
CacheLine Address	Offset	Thread Id	IBS_LOAD	IBS_LOAD	IBS_STORE	IBS_DC_MISS_LAT	IBS_LOAD_DC_L2_HIT	IBS_STORE_DC_MISS	IBS_NB	IBS_NF	IBS_NE	IBS_NB	IBS_NB	IBS_NB	Source File
0x3ecd4220c0	0x8	55896	6882	6882	0	38615	6540	0	0	0	342	342	0	0	0 read_func false_sharing_example.c 187
	0x10	55897	36757	0	36757	0	0	36655	0	0	0	0	0	0	0 write_func false_sharing_example.c 263
0xb617fa40c0	0x8	55896	182	182	0	1393	170	0	0	0	12	12	0	0	0 read_func false_sharing_example.c 187
	0x10	55897	1720	0	1720	0	0	1716	0	0	0	0	0	0	0 write_func false_sharing_example.c 263
0xb617fa4000	0x18	55896	4	4	0	0	4	0	0	0	0	0	0	0	0 read_func false_sharing_example.c 179
	0x18	55897	3	3	0	0	3	0	0	0	0	0	0	0	0 write_func false_sharing_example.c 247

Figure 38. Cache Analysis - Detailed Report

Use any of the following metric with the `--sort-by event=<METRIC>` (for example, `--sort-by event=ldst-count`) option to change the sorting by order during the report generation:

Table 37. Sort-by Metric

Sort-by Metric	Description
ldst-count	Total Loads and stores sampled
ld-count	Total Loads
st-count	Total Stores

Table 37. Sort-by Metric

Sort-by Metric	Description
cache-hitm	Loads that were serviced either from the local or remote cache (L3) and the cache hit state was Modified.
lcl-cache-hitm	Loads that were serviced from the local cache (L3) and the cache hit state was Modified
rmt-cache-hitm	Loads that were 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

Note: You can also use the command `info --list cacheline-events` for a list of supported metrics for `sort-by` option.

7.8 Custom Profile

Apart from the predefined configurations, you can choose the required events to profile. To perform the custom profile, complete the following steps:

7.8.1 Configuring and Starting Profile

1. Click **PROFILE > Start Profiling** to navigate to the **Select Profile Target** screen.
2. Selecting the required profile target and click the **Next** button.

The **Select Profile Type** screen is displayed.

- From the **Select Profile Type** drop-down, select **CPU Profile** as follows:

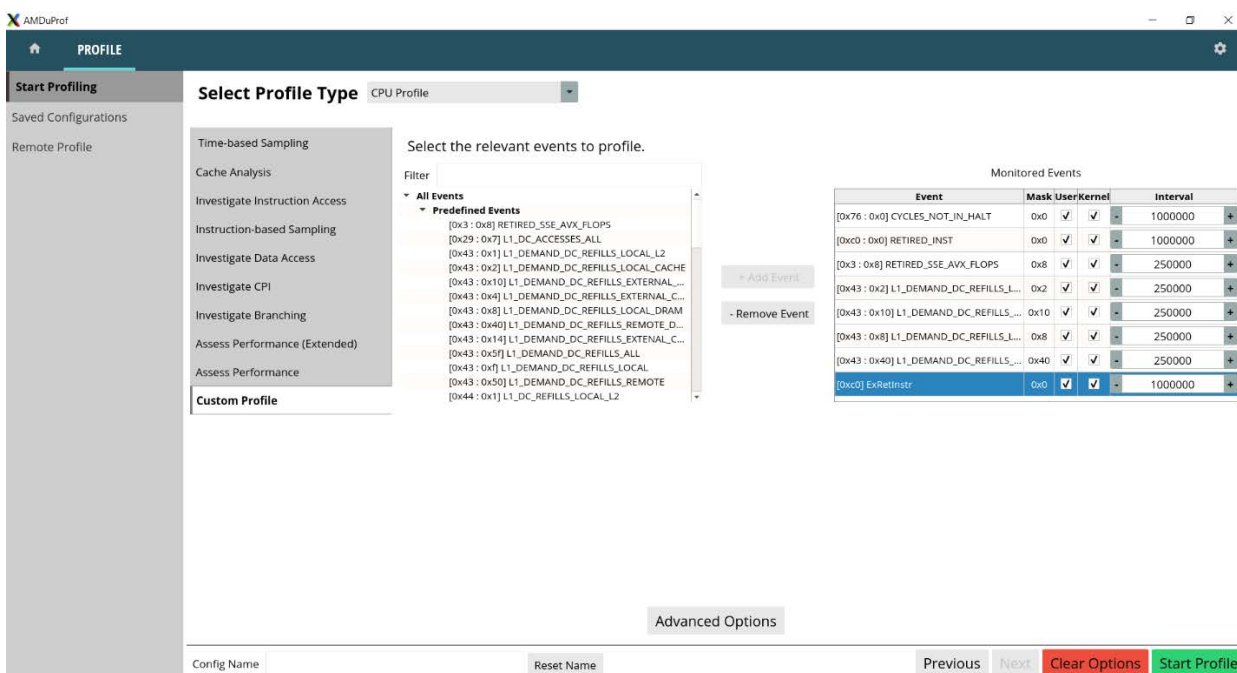


Figure 39. Start Profiling - Custom Profile

- Select **Custom Profile** in the left vertical pane.
- Click **Advanced Options** to enable call-stack, set symbol paths (if the debug files are in different locations) and other options. Refer the section “Advanced Options” on page 67 for more information on this screen.
- Once all the options are set, the **Start Profile** button at the bottom will be enabled. Click it to start the profile.

After the profile initialization the profile data collection screen is displayed.

7.8.2 Analyzing Profile Data

Complete the following steps to analyze the profile data:

- When the profiling stops, the collected raw profile data will be processed automatically and the **Hot Spots** screen of the **Summary** page is displayed. Refer the section “Overview of Performance Hotspots” on page 70 for more information on this screen.
- Click **ANALYZE** on the top horizontal navigation bar to go to the **Function HotSpots** screen. Refer the section “Function HotSpots” on page 71 for more information on this screen.
- Click **ANALYZE > Metrics** to display the profile data table at various granularities - Process, Load Modules, Threads, and Functions. Refer the section “Process and Functions” on page 73 for more information on this screen.

- Double-click any entry on the **Functions** table in **Metrics** screen to load the source tab for that function in **SOURCES** page. Refer the section “Source and Assembly” on page 75 for more information on this screen.

7.9 Advisory

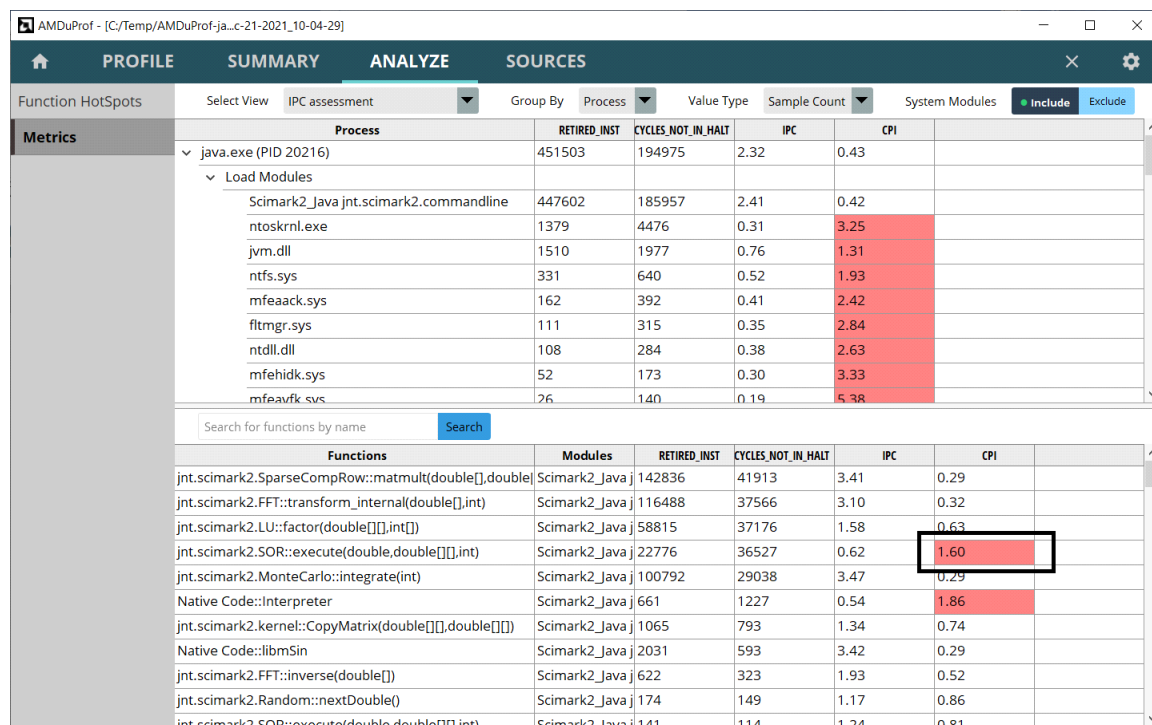
7.9.1 Confidence Threshold

The metric with low number of samples collected for a program unit either due to multiplexing or statical sampling will be grayed out. A few points to remember are:

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

7.9.2 Issue Threshold

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



Process	RETIRED_INST	CYCLES_NOT_IN_HALT	IPC	CPI
java.exe (PID 20216)	451503	194975	2.32	0.43
Load Modules				
Scimark2_Java jnt.scimark2.commandline	447602	185957	2.41	0.42
ntoskrnl.exe	1379	4476	0.31	3.25
jvm.dll	1510	1977	0.76	1.31
ntfs.sys	331	640	0.52	1.93
mfeaaack.sys	162	392	0.41	2.42
fltmgr.sys	111	315	0.35	2.84
ntdll.dll	108	284	0.38	2.63
mfehdk.sys	52	173	0.30	3.33
mfeavfk.sys	26	140	0.19	5.38

Functions	Modules	RETIRED_INST	CYCLES_NOT_IN_HALT	IPC	CPI
jnt.scimark2.SparseCompRow::matmult(double[], double[])	Scimark2_Java j	142836	41913	3.41	0.29
jnt.scimark2.FFT::transform_internal(double[], int)	Scimark2_Java j	116488	37566	3.10	0.32
jnt.scimark2.LU::factor(double[], int)	Scimark2_Java j	58815	37176	1.58	0.63
jnt.scimark2.SOR::execute(double, double[], int)	Scimark2_Java j	22776	36527	0.62	1.60
jnt.scimark2.MonteCarlo::integrate(int)	Scimark2_Java j	100792	29038	3.47	0.29
Native Code::Interpreter	Scimark2_Java j	661	1227	0.54	1.86
jnt.scimark2.kernel::CopyMatrix(double[], double[])	Scimark2_Java j	1065	793	1.34	0.74
Native Code::libmSin	Scimark2_Java j	2031	593	3.42	0.29
jnt.scimark2.FFT::inverse(double[])	Scimark2_Java j	622	323	1.93	0.52
jnt.scimark2.Random::nextDouble()	Scimark2_Java j	174	149	1.17	0.86
jnt.scimark2.SOR::execute(double, double[], int)	Scimark2_Java j	141	114	1.24	0.81

Figure 40. CPI Metric - Threshold-based Performance Issue

7.10 ASCII Dump of IBS Samples

For some scenarios, it would be useful to analyze the ASCII dump of IBS OP profile samples. To do so, complete the following 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\
```

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\AMDuProf-IBS_<timestamp>\
```

The CSV file that containing ASCII dump of the IBS OP samples is generated:

```
C:\temp\AMDuProf-IBS_<timestamp>\IbsOpDump.csv
```

3. During collection the following control knobs are available:

```
-e event=ibs-op,interval=100000,loadstore,ibsop-count-control=1
```

Where:

- interval denotes sampling interval
- loadstore denotes collect only the load & store ops (Windows only option)
- ibsop-count-control=1 represents count dispatched micro-ops (0 for “count clock cycles”)
- --data-buffer-count 1024 represents the number of per-core data buffers to allocate (Windows only option)

In case, there are too many missing records, try the following:

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

7.11 Limitations

CPU profiling in AMD uProf has the following limitations:

- CPU profiling expects the profiled application executable binaries must not be compressed or obfuscated by any software protector tools, for example, VMProtect.
- 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 8 Performance Analysis (Linux)

This chapter explains the Linux specific performance analysis models.

8.1 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. 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, the threads wait without doing anything useful due to several reasons:

- **Idle:** A thread finishes its 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:** The thread management overhead.

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

Support Matrix

The following table shows the support matrix:

Table 38. Support Matrix

Component	Supported Versions	Languages
OpenMP Spec	OpenMP v5.0	
Compiler	LLVM 8, 9, 10, 11, 12, 13, and 14	C and C++
	AOCC 2.1, 2.2, 2.3, 3.0, 3.1, 3.2, and 4.0	C, C++, and Fortran
	ICC 19.1 and 2021.1.1	C, C++, and Fortran
OS	Ubuntu 18.04 LTS, 20.04 LTS, and 22.04 LTS	
	RHEL 8.6 and 9	
	CentOS 8.4	

Prerequisite

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

8.1.1 Profiling OpenMP Application using GUI

Configuring and Starting a Profile

Complete the following steps to enable the OpenMP profiling:

1. Select the profile target and profile type.
2. Click the **Advanced Options** button.
3. In *Enable OpenMP Tracing* pane, turn on the **Enable OpenMP Tracing** option in, as shown in the following image:

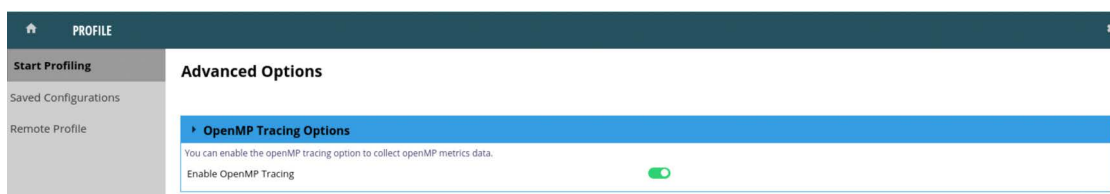


Figure 41. Enable OpenMP Tracing

Analyzing the OpenMP Report

After the profile completion, navigate to the *HPC* page to analyze the OpenMP tracing data. You can use the left side vertical pane on this page to navigate through the following views:

- **Overview** shows the quick details about the runtime. The following image shows the *Overview* page:

	PROFILE	SUMMARY	ANALYZE	HPC
Parallel Regions	OpenMP Summary			
Overview				
	Total Time :	0.94 s		
	Total Parallel Time :	0.91 s		
	Total Serial Time :	0.02 s		
	Parallel Time % :	97.53		
	Max Cores Utilized :	32		
	Total Threads Created :	32		

Figure 42. HPC - Overview

- **Parallel Regions** shows 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.

Parallel Region	Imbalance Time	Imbalance Time	Threads	Avg Idle Time (s)	Avg Sync Time (s)	Avg Overhead (s)	Avg Work Time (s)	Loop Chunk Size	Schedule Type	Elapsed Time (s)
VectorizedCellP	0.000000	0.000000	256	0.000000	0.000000	0.000008	0.000461	1	Static	0.000679
ParticlePairs2P	0.000000	0.000000	256	0.000000	0.000000	0.000009	0.000660	1	Static	0.001082
SlicedCellPairTr	0.000000	0.000000	256	0.000000	0.000000	0.000023	0.001294	1	Static	0.003590
LinkedCells:get	0.000000	0.000000	256	0.000000	0.000297	0.000022	0.000854	1	Static	0.001310
LinkedCells:get	0.000000	0.000000	256	0.000000	0.000235	0.000013	0.000667	1	Static	0.001025
VelocityScaling	0.000000	0.000000	256	0.000000	0.000000	0.000003	0.000339	1	Static	0.000553
Leapfrog:transl	0.000000	0.000000	256	0.000000	0.000000	0.000001	0.001006	1	Static	0.001571
LinkedCells:del	0.000000	0.000000	256	0.000000	0.000000	0.000173	0.000379	1	Static	0.000552
Simulation:upd	0.000000	0.000000	256	0.000000	0.000000	0.004387	0.006565	1	Static	0.010952
Simulation:upd	0.000000	0.000000	256	0.000000	0.000323	0.000165	0.000951	1	Static	0.001440
VectorizedCellP	0.000000	0.000000	256	0.000000	0.000524	0.000308	0.000985	1	Static	0.001817
CBSCellPairTrav	0.000000	0.000000	256	0.000000	0.003310	0.000248	3.119357	1	Static	3.122916
LinkedCells:sum	0.000000	0.000000	256	0.000000	0.000000	0.005464	0.021891	1	Static	0.027355

Thread no	Thread Id	Idle Time (secs)	Sync Time (secs)	Overhead Time (secs)	Work Time (secs)
0	130067	0.000000	0.000000	0.000028	0.000458
1	130073	0.000000	0.000000	0.000003	0.000556
2	130074	0.000000	0.000000	0.000002	0.000298
3	130075	0.000000	0.000000	0.000001	0.000199
4	130076	0.000000	0.000000	0.000004	0.000267
5	130077	0.000000	0.000000	0.000003	0.000599
6	130078	0.000000	0.000000	0.000001	0.000482
7	130079	0.000000	0.000000	0.000001	0.000288
8	130080	0.000000	0.000000	0.000001	0.000384
9	130081	0.000000	0.000000	0.000002	0.000669
10	130082	0.000000	0.000000	0.000002	0.000209

Figure 43. HPC - Parallel Regions

8.1.2 Profiling OpenMP Application Using CLI

Collect Profile Data

Use the following command to profile an OpenMP application using AMD uProf CLI:

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

While performing the regular profiling, add option `--trace openmp` or `--omp` to enable OpenMP profiling. This command will launch the program and collect the profile data required to generate the OpenMP analysis report.

Generate Profile Report

You can generate a CSV report using the `AMDuProfCLI report` command. Any additional option is not required for the OpenMP report generation. AMD uProf checks for the availability of any OpenMP profiling data and includes it in the report if available.

The following command will generate a CSV report in `/tmp/myapp_perf/<SESSION-DIR>/report.csv`:

```
$ ./AMDuProfCLI report -i /tmp/myapp_perf/<SESSION-DIR>
```

An example of the OpenMP report section in the CSV file is as follows:

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 Time	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	0.064491	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	0.060944	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	0.060453	0.409315					

Figure 44. An OpenMP Report

It has following sub-sections:

- **OpenMP OVERVIEW**
- **OpenMP PARALLEL-REGION METRIC** helps in understanding the imbalanced region, that is, a region with less total work time with respect to its total time. It has the following columns:
 - **Imbalance Time:** Total idle time spent by all the threads of the parallel region, normalized by the number of threads.
 - **Imbalance Time (%):** Percentage of the imbalance time with respect to the total time spent in the parallel region.
 - **Threads:** Number of threads in the parallel region.
 - **Avg Idle Time:** Average time spent by the parallel region threads waiting at the barrier for other threads to complete.
 - **Avg Sync Time:** Average time spent by the parallel region threads waiting on the synchronization locks to acquire the shared resource.
 - **Avg Overhead Time:** The thread management overhead.
 - **Avg Work Time:** Average time spent by the parallel region threads working.
 - **Loop Chunk Size:** Number of loop iterations scheduled for a chunk.
 - **Schedule Type:** Specifies how iterations of associated loops are divided into chunks and how these chunks are distributed among threads.
 - **Elapsed Time:** Time spent in the parallel region.

- **OpenMP THREAD METRIC** helps in understanding how each thread spent its time in the parallel region. If a thread spends too much time on non-work activity, the parallel region should be optimized further to improve the work time of each thread in that region. It has the following columns:
 - **ThreadNum**: Serial number of the thread.
 - **ThreadId**: Thread identifier.
 - **Idle Time**: Time spent by the thread waiting at the barrier for other threads to complete.
 - **Sync Time**: Time spent by the thread waiting on the synchronization locks to acquire the shared resource.
 - **Overhead Time**: Thread management overhead.
 - **Work Time**: Time spent by the thread working.

8.1.3 Environment Variables

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

8.1.4 Limitations

The following features not supported in this release:

- OpenMP profiling with system-wide profiling scope.
- Loop chunk size and schedule type when the parameters are specified using schedule clause. In such as case, it shows the default values (1 and Static).
- 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.
- Attaching to running OpenMP application.

8.2 MPI Profiling

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

1. Collect the profile data using CLI collect command.
2. Process the profile data using CLI translate command which will generate the profile database.

3. Import the profile database in the GUI or generate the CSV report using CLI report command.
4. Multiple ranks profiling requires higher limit to be set for memory locking using one of the following methods:
 - Increase the memory lock limit using the command `ulimit -l`, depending on the number of ranks to be profiled on the target node.
 - Set `/proc/sys/kernel/perf_event_paranoid` to -1 or higher value based on the profile config and scope.
 - Perform MPI profiling with root privilege.
5. Multiple ranks profiling might require a high number of file descriptors. If the file descriptor limit is reached during profile data collection, an error message will be displayed. You can increase this limit in the file `/etc/security/limits.conf`.
6. For Multiple ranks profiling, if the `/proc/sys/kernel/perf_event_paranoid` value is greater than -1, you must increase the `/proc/sys/kernel/perf_event_mlockb` value depending on the number of ranks to profile. Alternatively, you can also use the `-m` option to decrease the number of memory data buffer pages used by each instance of AMDuProfCLI.

Support Matrix

The MPI profiling supports the following components and the corresponding versions:

Table 39. MPI Profiling Support Matrix

Component	Supported Versions
MPI Spec	MPI v3.1
MPI Libraries	Open MPI v4.1.2
	MPICH v4.0.2
	ParaStation MPI v5.4.8
	Intel® MPI 2021.1
OS	Ubuntu 18.04 LTS, 20.04 LTS, and 22.04 LTS
	RHEL 8
	CentOS 8

8.2.1 Collecting Data Using CLI

The MPI jobs are launched using MPI launchers such as *mpirun* and *mpiexec*. You must use AMDuProfCLI to collect the profile data for an MPI application.

The MPI job launch through *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, use the following syntax to profile an MPI application using AMDuProfCLI:

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

The MPI profiling specific AMDuProfCLI options:

- The `--mpi` option 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. A session directory will be created within the `<output dir>` containing all the data collected from all the ranks.

A typical command uses the following syntax:

```
$ mpirun -np <n> /tmp/AMDuProf/bin/AMDuProfCLI collect
--config <config-type> --mpi --output-dir <output_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. You can either analyze the data for processes ran on one/many/all node(s).

Method 1 - Profile All the Ranks On Single/Multiple Node(s)

To collect profile data for all the ranks running on a single node, execute the following commands:

```
$ 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 the options `-H / --host mpirun` 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 Specific Rank(s)

To profile only a single rank running on host2, execute the following commands:

```
$ 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 1 $AMDUPROFCLI_CMD myapp.exe
```

To profile only a single rank in setup where 256 ranks running on 2 hosts (128 ranks per host):

```
$ mpirun -host host1:128 -np 1 $AMDUPROFCLI_CMD myapp.exe : -host host2:128,host1:128 -np 255
--map-by core myapp.exe
```

Method 3 – Using MPI Config File

The `mpirun` also takes config file as an input and the AMDuProfCLI can 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, execute the following command:

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

8.2.2 Analyzing the Data with CLI

The data collected for MPI processes can be analyzed using the CSV reported by the AMDuProfCLI report command. The generated report is saved to the *file report.csv* in the *<output-dir>/<SESSION-DIR>* folder.

Following are the reporting options for the CLI:

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

```
$ AMDuProfCLI report --input-dir /tmp/myapp-perf/<SESSION-DIR> --host host1
```

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

- Generate a report for all the MPI processes ran on another host (for example, host2) in which the MPI launcher was not launched:

```
$ AMDuProfCLI report --input-dir /tmp/myapp-perf/<SESSION-DIR> --host host2
```

- Generate a report for all the MPI processes ran on all the hosts:

```
$ AMDuProfCLI report --input-dir /tmp/myapp-perf/<SESSION-DIR> --host all
```

8.2.3 Analyze the Data with GUI

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

1. To generate the profile database, refer “Analyzing the Data with CLI” on page 137.
2. To import the profile database, refer “Importing Profile Database” on page 80.

8.2.4 Limitations

The MPI environment parameters such as **Total number of ranks** and **Number of ranks running on each node** are currently supported only for OpenMPI. MPI profiling with system-wide profiling scope is not supported.

8.3 Profiling Support on Linux for perf_event_paranoid Values

Following table describes profiling support on Linux for different perf_event_paranoid values:

Table 40. Profiling perf_event_paranoid Values on Linux

Config	Profile Scope	perf_event_paranoid Values			
		-1	0	1	2
Timer Based Profiling	Specific Application or Process	Y	Y	Y	Y
Timer Based Profiling	Kernel, Hypervisor	Y	Y	Y	N
Timer Based Profiling	Entire System	Y	Y	N	N
Core PMC Event Based Profiling	Specific Application or Process	Y	Y	Y	Y
Core PMC Event Based Profiling	Kernel, Hypervisor	Y	Y	Y	N
Core PMC Event Based Profiling	Entire System	Y	Y	N	N
Instruction Based Sampling	Specific Application or Process	Y	Y	N	N
Instruction Based Sampling	Entire System	Y	Y	N	N

8.4 Profiling Linux System Modules

To attribute the samples to the system modules (for example, glibc and libm), AMD uProf uses the corresponding debug info files. The Linux distros do not contain the debug info files, but most of the popular distros provide options to download the debug info files.

Refer the following resources for more information on how to download the debug info files:

- Ubuntu (<https://wiki.ubuntu.com/Debug%20Symbol%20Packages>)
- RHEL/CentOS (https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Developer_Guide/intro.debuginfo.html)

Ensure that you download the debug info files for the required system modules for the required Linux distros before starting the profiling.

8.5 Profiling Linux Kernel

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

1. Enable the kernel symbol resolution.

2. Do one of the following:

- Download and install kernel debug symbol packages and source.
- Build Linux kernel with debug symbols.

After the kernel debug info is available in the default path, AMD 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, and RHEL 8

8.5.1 Enabling Kernel Symbol Resolution

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

- Set `/proc/sys/kernel/perf_event_paranoid` to **-1**.
- Set `/proc/sys/kernel/kptr_restrict` to an appropriate value as follows:
 - **0**: The kernel addresses are available without any limitations.
 - **1**: The kernel addresses are available if the current user has a `CAP_SYSLOG` capability.
 - **2**: The kernel addresses are hidden.

Set the **perf_event_paranoid** value using one of the following:

```
$ sudo echo -1 > /proc/sys/kernel/perf_event_paranoid  
or  
$ sudo sysctl -w kernel.perf_event_paranoid=-1
```

Set the **kptr_restrict** value using one of the following:

```
$ sudo echo 0 > /proc/sys/kernel/kptr_restrict  
or  
$ sudo sysctl -w kernel.kptr_restrict=0
```

8.5.2 Downloading and Installing Kernel Debug Symbol Packages

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

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

Ubuntu

Complete the following steps to download kernel debug info and source code on Ubuntu systems (verified on Ubuntu 18.04.03 LTS):

1. To trust the debug symbol signing key, execute the following commands:

```
// Ubuntu 18.04 LTS and later:
$ sudo apt install ubuntu-dbgsym-keyring
// For earlier releases of Ubuntu:
$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
F2EDC64DC5AEE1F6B9C621F0C8CAB6595FDFF622
```

2. Add the debug symbol repository as follows:

```
$ 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)
```

After the kernel debug info file is downloaded, it can be found at the default path:

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

RHEL

Follow the steps in Red Hat knowledgebase (<https://access.redhat.com/solutions/9907>) to download the RHEL kernel debug info.

After the kernel debug info file is downloaded, it can be found at the default path:

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

8.5.3 Build Linux kernel with Debug Symbols

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

8.5.4 Analyzing Hotspots in Kernel Functions

If the debug info for the 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.

During the hotspot analysis, do consider the following:

- 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 in the GUI. The CLI should also be able to generate source level report and IMIX report for the kernel.
- If the source is downloaded and copied to the expected path, then you should be able to see the kernel source lines in GUI and CLI.
- Passing of kernel debug file path and passing of kernel source path is not recommended as that might lead to performance issues.

8.5.5 Linux Kernel Callstack Sampling

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

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

8.5.6 Constraints

- Do not move the downloaded kernel debug info from its default path.
- If the kernel version gets upgraded, then download the kernel debug info for the latest kernel version. AMD uProf would not show correct source and assembly if there is any mismatch between kernel debug info and kernel version.
- While profiling or analyzing kernel samples, do not reboot the system in between. Rebooting the system would cause the kernel to load at a different virtual address due to the KASLR feature of Linux kernel.
- The settings in the `/proc/sys/kernel/kptr_restrict` file enables AMD uProf to resolve kernel symbols and attribute samples to kernel functions. It does not enable the source and assembly level, call-graph analysis.

8.6 Holistic Analysis

The OS and runtime libraries can be traced along with CPU sampling-based profiles to provide timeline views to analyze what is happening in the system when that application is running. Following trace events can be collected and analyzed:

- OS scheduling event

- System calls
- POSIX threads library's (pthread) thread synchronization APIs
- API and GPU activity tracing for heterogeneous application using HIP
- MPI API event tracing

This tracing helps visualize:

- CPU thread states timeline – whether a thread is running, sleeping, or waiting for a resource to be available.
- Thread is running in kernel-space or user-space and if the thread is running in kernel-space, which system call is being called.
- If the thread is not scheduled to run, identify whether it is due to blocking I/O or waiting on synchronization object.
- If the profiled application is HIP based heterogeneous application, visualize the HIP & HSA APIs and GPU activities.
- MPI API timeline.

AMD uProf uses:

- Linux eBPF tracing framework to trace the OS and runtime events. Hence, OS Tracing option is supported only if eBPF is supported and BCC (BPF Compiler Collection) tool is installed on the host system. eBPF is supported in Linux kernel version 4.7 and later (it is recommended to use kernel 4.15 or later). Use the command `AMDuProfCLI info --bpf` to check whether BPF support is available on the system and BCC is installed.
- AMD ROCtracer library to trace the ROCr (HSA API) and HIP supported GPU activity.
- MPI API tracing is supported using PMPI interface for OpenMPI, MPICH, and their derivatives.

Supported Events

Use the command `AMDuProfCLI info --list ostrace-events` to list the supported events on the target system. AMD uProf supports the following system events for tracing to show timeline view in GUI:

Table 41. Supported Events for Holistic Analysis

Category	Event	Description
OS and Runtime	schedule	To trace the thread state and the core on which the thread is running.
OS and Runtime	syscall	To collect the time taken by the system calls and the count of each system call. By default, traces the system calls which are executing for >10 usec. To enable or disable specific system call tracing, update the config file located in: <i>AMDuProf_Linux_x64_X.Y.ZZZ/bin/bpf/AMDTSysCallList.json</i>
OS and Runtime	pthread	To trace POSIX thread (pthread) library synchronization APIs. It provides the reason for thread state change. <i>Note: Supported only for application level tracing.</i>

Table 41. Supported Events for Holistic Analysis

GPU	hip	HIP runtime trace.
GPU	hsa	AMD ROCr runtime trace.
MPI	mpi	MPI API trace.

Prerequisites

For tracing OS events and runtime libraries:

- Requires Linux kernel 4.7 or later (it is recommended to use kernel 4.15 or later).
- Root access is required to trace the OS events in Linux.
- To install BCC and eBPF scripts, refer section “Installing BCC and eBPF” on page 24. To validate the BCC Installation, run the script *sudo AMDuProfVerifyBpfInstallation.sh*.

For tracing HIP, HSA APIs, and GPU activity — Requires AMD ROCm 5.2.1 to be installed. For the steps to install AMD ROCm, refer section “Installing ROCm” on page 24.

For tracing MPI APIs — OpenMPI v4.1 or later and MPICH v3.4 runtime or later

8.6.1 Holistic Analysis Using CLI

The AMDuProfCLI can be used to collect the required trace data and generate the report in .csv format for further analysis. The processed profile data can also be imported in GUI.

Collect Profile Data

The CLI has an option called `--trace` to specify the OS events and runtime libraries to be traced.

Example CLI command to trace OS scheduling, system calls, and thread synchronization APIs along with time-based sampling for performing holistic analysis:

```
$ sudo AMDuProfCLI collect --config tbp -trace os -o /tmp/holistic-analysis/ /home/app/classic
...
Generated data files path: /tmp/holistic-analysis/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27
```

This command will launch the program and collect the profile and trace data. Once the launched application is executed, the AMDuProfCLI will display the session directory path in which the raw profile and trace data are saved.

In the above example, the session directory path is:

/tmp/holistic-analysis/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27/

For HIP based applications, example CLI command to trace OS scheduling, system calls, thread synchronization APIs, and GPU activity along with time-based sampling for performing holistic analysis:

```
$ sudo AMDuProfCLI collect --config tbp -trace os --trace gpu -o /tmp/holistic-analysis/ /home/app/classic
...
Generated data files path: /tmp/holistic-analysis/AMDuProf-classic-0sTrace_Dec-09-2021_12-19-27
```

Generate Profile Report

Use the following CLI report command to generate the profile report in .csv format by passing the session directory path as the argument to -i option:

```
$ ./AMDuProfCLI report -i /tmp/holistic-analysis/AMDuProf-classic-0sTrace_Dec-09-2021_12-19-27
...
Generated report file: /tmp/holistic-analysis/AMDuProf-classic-0sTrace_Dec-09-2021_12-19-27/report.csv
```

After processing the data and generating the report, the report file path is displayed on the terminal. An example of the OS trace report section in the .csv report file is as follows:

OS TRACING REPORT						
SYSTEM CALL SUMMARY						
System Call	Count	Total Elapsed Tim	Min Elapsed Ti	Max Elapsed T	Avg Elapsed Time(seconds)	
futex	11	194.28	1.00E-06	58.2367	17.6618	
nanosleep	1	10.0002	10.0002	10.0002	10.0002	
THREAD SUMMARY						
Process	Thread	Elapsed Time(sec	Wait Time(sec	Wait Time(% From Thread Elapsed Time)		
/home/amd/testApplication/classic_lock(223170)	"classic_lock(223170)"	88.1247	88.1106	99.98		
/home/amd/testApplication/classic_lock(223170)	"MatrixMulti(223324)"	77.7414	58.2367	74.91		
/home/amd/testApplication/classic_lock(223170)	"MatrixMulti(223323)"	58.2412	38.8395	66.69		
/home/amd/testApplication/classic_lock(223170)	"MatrixMulti(223322)"	38.8425	19.4699	50.13		
/home/amd/testApplication/classic_lock(223170)	"MatrixMulti(223321)"	19.4726	0.000243567	0		
WAIT OBJECT SUMMARY						
Wait Object	Thread	Wait Count	Total Wait Tim	Wait Time (% From Object Total Wait Time)		
Mutex@0x5596d20f8080	"MatrixMulti(223324)"	1	58.2368	49.970056		
Mutex@0x5596d20f8080	"MatrixMulti(223323)"	1	38.8385	33.325341		
Mutex@0x5596d20f8080	"MatrixMulti(223322)"	1	19.4681	16.704599		
Mutex@0x5596d20f8080	"MatrixMulti(223321)"	1	5.10E-06	0.000004		
Mutex@0x7f76a47ec968	"classic_lock(223170)"	2	4.49E-06	100		
WAIT FUNCTION SUMMARY						
Function	Wait Count	Total Wait Time(seconds)				
pthread_mutex_lock@0x5596d20f8080	4	116.543				
pthread_mutex_unlock@0x5596d20f8080	4	3.84E-05				
pthread_mutex_lock@0x7f76a47ec968	2	4.49E-06				
pthread_mutex_unlock@0x7f76a47ec968	2	3.73E-06				

Figure 45. System Analysis - OS Tracing Report

Analyze Trace Data with GUI

To visualize the trace data collected using CLI, the collected raw profile and trace data should be processed using CLI translate command and then it can be imported in the GUI. This will plot the timecharts and other views in the GUI using the profile and trace data collected.

Use the following CLI translate command invocation to process the raw trace records saved in the corresponding session directory path:

```
$ ./AMDuProfCLI translate -i /tmp/holistic-analysis/AMDuProf-classic-OSTrace_Dec-09-2021_12-19-27
...
Translation finished
```

Then import this session in the GUI by specifying the session directory path in **Profile Data File** text input box in the **HOME > Import Session** view. This will load the profile data saved in the session directory for further analysis.

Navigate to the **TIMECHART** page and select **System Analysis** in the **Timeline Category** drop-down. This view shows the timeline of combined OS and Runtime tracing and GPU activity tracing as follows:

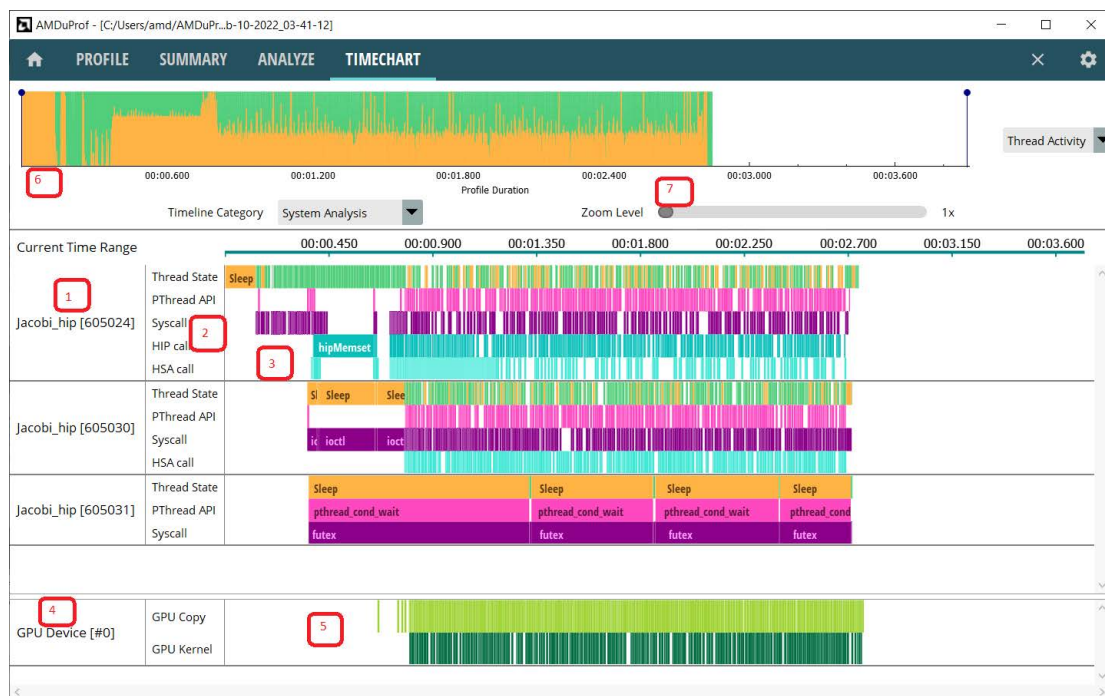


Figure 46. System Analysis - TIMECHART

In the above figure:

1. The process or thread name along with the thread ID.
2. The OS and runtime characteristics for which tracing data is available for the current thread.
3. Timeline swim lanes for the various OS and runtime characteristics for which tracing data is available.

If huge number of samples are collected for an OS or runtime event, then selective samples (selected based on adaptive sampling approach) will be plotted. Either by increasing the zooming using **Zoom Level** slider or by performing selective filtering by selecting interesting time range will load more trace samples for analysis.

4. The GPU device(s) on which the kernels were running.
5. The GPU activities on this GPU device – GPU Copy and GPU Kernel executions.
6. Thread or GPU Activity with time range selector – Select the time range so that the GUI plots the trace samples within the selected time range. Use the drop-down to select **Thread** or **GPU Activity**.
7. To zoom in or zoom out, use the **Zoom Level** slider. The zoom in will fetch and render more trace samples.

8.7 Thread State Analysis

The OS and runtime libraries can be traced along with CPU sampling-based profiles to visualize the thread synchronization of a multi-threaded application and the threads core affinity. Following trace events should be collected for this analysis:

- OS scheduling event
- System calls
- POSIX threads library (pthread) thread synchronization APIs

This tracing helps visualize:

- CPU thread states timeline – whether a thread is running, sleeping, or waiting for a resource to be available.
- Thread is running in kernel-space or user-space and if the thread is running in kernel-space, the system call being called.
- If the thread is not scheduled to run, whether it is blocked due to I/O or waiting for a synchronization object.

Prerequisites

For tracing OS events and runtime libraries:

- Requires Linux kernel 4.7 or later (it is recommended to use kernel 4.15 or later).
- Root access is required to trace the OS events in Linux.
- To install BCC and eBPF scripts, refer section “Installing BCC and eBPF” on page 24. To validate the BCC Installation, run the script `sudo AMDuProfVerifyBpfInstallation.sh`.

8.7.1 Thread State Analysis Using CLI

The AMDuProfCLI can be used to collect the required trace data and generate the report in .csv format for further analysis. The processed profile data can also be imported in GUI.

Collect Profile Data

The CLI has an option called `--trace` to specify the OS events and runtime libraries to be traced.

Example CLI command to trace OS scheduling, system calls, and POSIX thread synchronization APIs along with time-based sampling for performing holistic analysis:

```
$ sudo AMDuProfCLI collect --config tbp -trace os -o /tmp/holistic-analysis/ /home/app/classic
...
Generated data files path: /tmp/holistic-analysis/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27
```

This command will launch the program and collect the profile and trace data. Once the launched application is executed, the AMDuProfCLI will display the session directory path in which the raw profile and trace data are saved.

In the above example, the session directory path is:

```
/tmp/holistic-analysis/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27/
```

Generate Profile Report

Use the following CLI report command to generate the profile report in .csv format by passing the session directory path as the argument to -i option:

```
$ ./AMDuProfCLI report -i /tmp/thread-analysis/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27
...
Generated report file: /tmp/thread-analysis/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27/
report.csv
```

After processing the data and generating the report, the report file path is displayed on the terminal. An example of the OS trace report section in the .csv report file is as follows:

OS TRACING REPORT														
SYSTEM CALL SUMMARY														
System Call	Count	Total Elapsed Time	Min Elapsed Time	Max Elapsed Time	Avg Elapsed Time(seconds)									
futex	11	194.28	1.00E-06	58.2367	17.6618									
nanosleep	1	10.0002	10.0002	10.0002	10.0002									
THREAD SUMMARY														
Process	Thread	Elapsed Time(seconds)	Wait Time(seconds)	Wait Time(% From Thread Elapsed Time)										
/home/amd/testApplication/classic_lock(223170)	"classic_lock(223170)"	88.1247	88.1106	99.98										
/home/amd/testApplication/classic_lock(223170)	"MatrixMulti(223324)"	77.7414	58.2367	74.91										
/home/amd/testApplication/classic_lock(223170)	"MatrixMulti(223323)"	58.2412	38.8395	66.69										
/home/amd/testApplication/classic_lock(223170)	"MatrixMulti(223322)"	38.8425	19.4699	50.13										
/home/amd/testApplication/classic_lock(223170)	"MatrixMulti(223321)"	19.4726	0.000243567	0										
WAIT OBJECT SUMMARY														
Wait Object	Thread	Wait Count	Total Wait Time	Wait Time (% From Object Total Wait Time)										
Mutex@0x5596d20f8080	"MatrixMulti(223324)"	1	58.2368	49.970056										
Mutex@0x5596d20f8080	"MatrixMulti(223323)"	1	38.8385	33.325341										
Mutex@0x5596d20f8080	"MatrixMulti(223322)"	1	19.4681	16.704599										
Mutex@0x5596d20f8080	"MatrixMulti(223321)"	1	5.10E-06	0.000004										
Mutex@0x7f76a47ec968	"classic_lock(223170)"	2	4.49E-06	100										
WAIT FUNCTION SUMMARY														
Function	Wait Count	Total Wait Time(seconds)												
pthread_mutex_lock@0x5596d20f8080	4	116.543												
pthread_mutex_unlock@0x5596d20f8080	4	3.84E-05												
pthread_mutex_lock@0x7f76a47ec968	2	4.49E-06												
pthread_mutex_unlock@0x7f76a47ec968	2	3.73E-06												

Figure 47. Thread State Analysis - OS Tracing Report

Analyze Trace Data with GUI

To visualize the trace data collected using CLI, the collected raw profile and trace data should be processed using CLI translate command and then it can be imported in the GUI. This will plot the timecharts and other views in the GUI using the profile and trace data collected.

Use the following CLI translate command invocation to process the raw trace records saved in the corresponding session directory path:

```
$ ./AMDuProfCLI translate -i /tmp/thread-analysis/AMDuProf-classic-0sTrace_Dec-09-2021_12-19-27
...
Translation finished
```

Then import this session in the GUI by specifying the session directory path in **Profile Data File** text input box in the **HOME > Import Session** view. This will load the profile data saved in the session directory for further analysis.

Navigate to the **TIMECHART** page and select **Thread State Analysis** in the **Timeline Category** drop-down. This view shows the timeline of combined OS and runtime libraries events traced as follows:

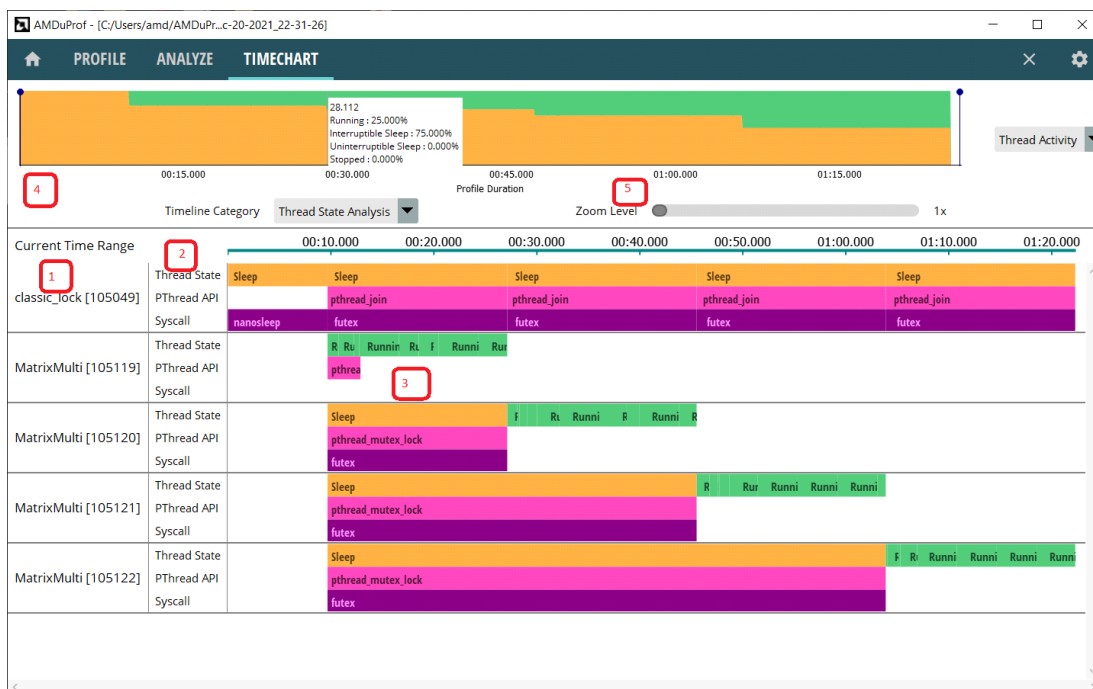


Figure 48. Thread State Analysis - TIMECHART

In the above figure:

1. The process or thread name along with the thread ID.
2. The OS and runtime characteristics for which tracing data is available for the current thread.

3. Timeline swim lanes for the various OS and runtime characteristics for which tracing data is available.

If huge number of samples are collected for an OS or runtime event, then selective samples (selected based on adaptive sampling approach) will be plotted. Either by increasing the zooming using **Zoom Level** slider or by performing selective filtering by selecting interesting time range will load more trace samples for analysis.

4. Thread Activity with time range selector – Select the time range so that GUI will only plot the trace samples within the selected time range.
5. To zoom in or zoom out, use the **Zoom Level** slider. The zoom in will fetch and render more trace samples.

8.8 Kernel Block I/O Analysis

The Linux OS block I/O calls like insert, issue, and complete can be traced to provide the various metrics related to I/O operations performed by the application.

Table 42. I/O Operations

Category	Event	Description
OS and Runtime	diskio	To trace the block I/O operations when the application is running.

This analysis can be used to analyze:

- Time taken to complete the I/O operations
- IOPS - Number of block I/O operations per second
- Read or Write bytes of block I/O operation
- Block I/O bandwidth

Note: The kernel can continue to perform the queued I/O requests submitted by the profiled application, even after the application exits. So, it is recommended to use system-wide tracing for this analysis.

Prerequisites

For tracing OS events and runtime libraries:

- Requires Linux kernel 4.7 or later (it is recommended to use kernel 4.15 or later).
- Root access is required to trace the OS events in Linux.
- To install BCC and eBPF scripts, refer section “Installing BCC and eBPF” on page 24. To validate the BCC Installation, run the script `sudo AMDuProfVerifyBpfInstallation.sh`.

8.8.1 Kernel Block I/O Analysis Using CLI

The AMDuProfCLI can be used to collect the required trace data and generate the report in .csv format for further analysis. The processed profile data can also be imported in GUI.

Collect Profile Data

Example CLI command to trace block I/O operations along with time-based sampling:

```
$ sudo AMDuProfCLI collect --config tbp -trace os=diskio -o /tmp/blockio-analysis/ /usr/bin/
fio ...
...
Generated data files path: /tmp/blockio-analysis/AMDuProf-fio-OsTrace_Dec-09-2021_12-19-27
```

This command will launch the program and collect the profile and trace data. Once the launched application is executed, the AMDuProfCLI will display the session directory path in which the raw profile and trace data are saved.

In the above example, the session directory path is:

/tmp/blockio-analysis/AMDuProf-fio-OsTrace_Dec-09-2021_12-19-27/

Generate Profile Report

Use the following CLI report command to generate the profile report in .csv format by passing the session directory path as the argument to -i option:

```
$ ./AMDuProfCLI report -i /tmp/blockio-analysis/AMDuProf-fio-OsTrace_Dec-09-2021_12-19-27
...
Generated report file: /tmp/blockio-analysis/AMDuProf-fio-OsTrace_Dec-09-2021_12-19-27/
report.csv
```

After processing the data and generating the report, the report file path is displayed on the terminal. An example of the disk I/O report section in the .csv report file is as follows:

MONITORED EVENTS										
OS Trace Events:		Name	Threshold	Description						
		DISKIO	0	Disk I/O tracing						
OS TRACING REPORT										
DISK IO SUMMARY										
Device	Access Count	IOPS	Total Read	Total Write	Total Read S	Total Write S	Avg IO Latenc	Read Bandwidth	Write Bandwidth	(MBPS)
/dev/nvme0n1	24672	498	25	24647	0.1024	25797.4	220.256	0.00206889	521.213	
/dev/sda	150	3	18	127	3.31776	2.77299	4.50939	0.0674576	0.056381	
/dev/sdb	5	384	0	0	0	0	2.52706	0	0	
DISK IO SUMMARY (PROCESS)										
Process	Device	Access Count	IOPS	Total Read C	Total Write	Total Read S	Total Write S	Avg IO Latency(msec)		
/usr/bin/fio(102146)	"/dev/nvme0n1"	25	907	25	0	0.1024	0	0.321129		
/usr/bin/fio(102146)	"/dev/sda"	18	0	18	0	3.31776	0	23.9266		

Figure 49. Disk I/O Summary Tables

Analyze Trace Data with GUI

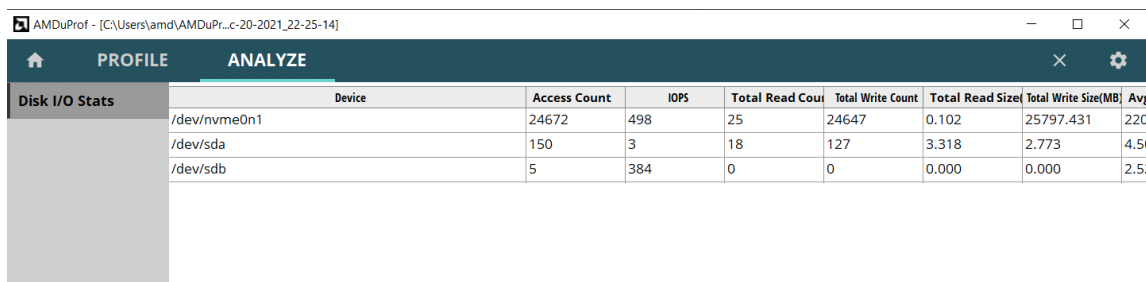
To visualize the trace data collected using CLI, the collected raw profile and trace data should be processed using CLI translate command and then it can be imported in the GUI.

Use the following CLI translate command invocation to process the raw trace records saved in the corresponding session directory path:

```
$ ./AMDuProfCLI translate -i /tmp/blockio-analysis/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-
27
...
Translation finished
```

Then import this session in the GUI by specifying the session directory path in **Profile Data File** text input box in the **HOME > Import Session** view. This will load the profile data saved in the session directory for further analysis.

Navigate to the **ANALYZE** page and then select **Disk I/O Stats** in the vertical navigation bar as follows:



Device	Access Count	IOPS	Total Read Count	Total Write Count	Total Read Size	Total Write Size (MB)	Avg
/dev/nvme0n1	24672	498	25	24647	0.102	25797.431	220
/dev/sda	150	3	18	127	3.318	2.773	4.50
/dev/sdb	5	384	0	0	0.000	0.000	2.50

Figure 50. ANALYZE - Block I/O Stats

In the above figure, the table shows various block I/O statistics at the device level.

8.9 GPU Offloading Analysis (GPU Tracing)

GPU offloading analysis is used to explore the traces of the function calls for a GPU compute-intensive application.

The AMD ROCtracer library provides support to capture the runtime APIs and GPU activities such as data transfer and kernel execution. This analysis helps to visualize the ROCr, HIP API calls, and GPU activities when a HIP based application is running. It is supported only with a launch application.

Supported Interfaces

AMD uProf supports tracing the following ROCr runtime APIs, GPU activities, and to show the data in GUI timeline view:

Table 43. Supported Interfaces for GPU Tracing

Category	Event	Description
GPU	hip	HIP runtime trace
GPU	hsa	AMD ROCr runtime trace

Prerequisites

For tracing ROCr, HIP APIs, and GPU activities:

- Requires AMD ROCm 5.2.1 to be installed. For the steps to install AMD ROCm, refer section “Installing ROCm” on page 24.

Note: Tracing might not work as expected on '5.1.3 or older' and '5.2.3 or later' versions.

- Support accelerators - AMD Instinct™ MI100 and MI200

Optional Settings

By default, AMDuProf uses the:

- ROCm version pointed by `/opt/rocm/` symbolic link. To specify the rocm path, you must export it using `AMDUPROF_ROCM_PATH` before launching AMD uProf.

Example:

```
export AMDUPROF_ROCM_PATH=/opt/rocm-5.2.1/
```

- ROCm libraries from `/opt/rocm/lib`. If `AMDUPROF_ROCM_PATH` is specified, the specified path or library will be used. To change this path, you must export it using `AMDUPROF_ROCM_LIB_PATH` before launching AMD uProf.

Example:

```
export AMDUPROF_ROCM_LIB_PATH=/opt/rocm-5.2.1/lib
```

8.9.1 GPU Offload Analysis Using CLI

The `AMDuProfCLI` can be used to collect the required trace data and generate the report in `.csv` format for further analysis. The processed profile data can also be imported in GUI.

Collect Profile Data

The CLI has an option `--trace` to specify the GPU events and runtime libraries to be traced. For HIP based applications, example CLI command to trace ROCr, HIP APIs, and GPU activity along with time-based sampling for performing GPU offload analysis:

```
$ sudo AMDuProfCLI collect --config tbp --trace gpu -o /tmp/gpu-analysis/ /home/app/SampleApp
...
Generated data files path: /tmp/gpu-analysis/AMDuProf-SampleApp-GpuTrace_Dec-09-2021_12-19-27
```

This command will launch the program and collect the profile and trace data. Once the launched application is executed, the `AMDuProfCLI` will display the session directory path in which the raw profile and trace data are saved.

In the above example, the session directory path is:

```
/tmp/gpu-analysis/AMDuProf-SampleApp-GpuTrace_Dec-09-2021_12-19-27/
```

Generate Profile Report

Use the following CLI report command to generate the profile report in `.csv` format by passing the session directory path as the argument to `-i` option:

```
$ ./AMDuProfCLI report -i /tmp/gpu-analysis/AMDuProf-SampleApp-GpuTrace_Dec-09-2021_12-19-27
...
Generated report file: /tmp/gpu-analysis/AMDuProf-SampleApp-OsTrace_Dec-09-2021_12-19-27/
report.csv
```

After processing the data and generating the report, the report file path is displayed on the terminal. An example of the GPU trace report section in the `.csv` report file is as follows:

GPU TRACING REPORT					
KERNEL SUMMARY					
Name	Count	Elapsed Time(second)	Avg Elapsed Time	Elapsed Time(% From Total API Elapsed Time)	
JacobilerationKernel(int, double, dc	1000	0.549017	0.000549017	41.9465	
NormKernel1(int, double, double, do	1001	0.470506	0.000470036	35.948	
LocalLaplacianKernel(int, int, int, do	1000	0.270641	0.000270641	20.6777	
HaloLaplacianKernel(int, int, int, dou	1000	0.015341	1.53E-05	1.1721	
NormKernel2(int, double const*, do	1001	0.00334609	3.34E-06	0.255651	
DATA TRANSFER SUMMARY					
Name	Count	Elapsed Time(second)	Avg Elapsed Time	Elapsed Time(% From Total API Elapsed Time)	
CopyHostToDevice	4	0.00478305	0.00119576	54.4825	
CopyDeviceToHost	1001	0.00398993	3.99E-06	45.4482	
FillBuffer	1	6.08E-06	6.08E-06	0.0692557	
HIP API SUMMARY					
Name	Count	Elapsed Time(second)	Avg Elapsed Time	Elapsed Time(% From Total API Elapsed Time)	
hipMemcpy	1005	0.920585	0.000916005	54.4886	
hipLaunchKernel	5002	0.282263	5.64E-05	16.7069	
hipMemset	1	0.266986	0.266986	15.8026	
hipEventRecord	2000	0.143416	7.17E-05	8.48868	
hipEventElapsedTime	1000	0.0270553	2.71E-05	1.60138	
hipStreamCreate	2	0.0174582	0.00872911	1.03334	
hipDeviceSynchronize	1001	0.0161897	1.62E-05	0.958252	
__hipPushCallConfiguration	5002	0.00542041	1.08E-06	0.320829	
__hipPopCallConfiguration	5002	0.00519112	1.04E-06	0.307257	
hipStreamSynchronize	2000	0.00242338	1.21E-06	0.143438	
HSA API SUMMARY					
Name	Count	Elapsed Time(second)	Avg Elapsed Time	Elapsed Time(% From Total API Elapsed Time)	
hsa_signal_wait_scacquire	4035	0.346774	8.59E-05	55.7946	
hsa_system_get_info	44090	0.0470886	1.07E-06	7.57638	
hsa_amd_profiling_get_dispatch_tin	7003	0.0404551	5.78E-06	6.50908	

Figure 51. GPU Tracing Report

Analyze Trace Data with GUI

To visualize the trace data collected using CLI, the collected raw profile and trace data should be processed using CLI translate command and then it can be imported in the GUI. This will plot the timecharts and other views in the GUI for the collected profile and trace categories.

Use the following CLI translate command invocation to process the raw trace records saved in the corresponding session directory path:

```
$ ./AMDuProfCLI translate -i /tmp/gpu-analysis/AMDuProf-SampleApp-GpuTrace_Dec-09-2021_12-19-27
...
Translation finished
```

Then import this session in the GUI by specifying the session directory path in **Profile Data File** text input box in the **HOME > Import Session** view. This will load the profile data saved in the session directory for further analysis.

Navigate to the **TIMECHART** page and then select **GPU Offload Analysis** in the **Timeline Category** drop-down. This view shows the timeline of ROCr, HIP runtime APIs tracing, and GPU activity tracing as follows:

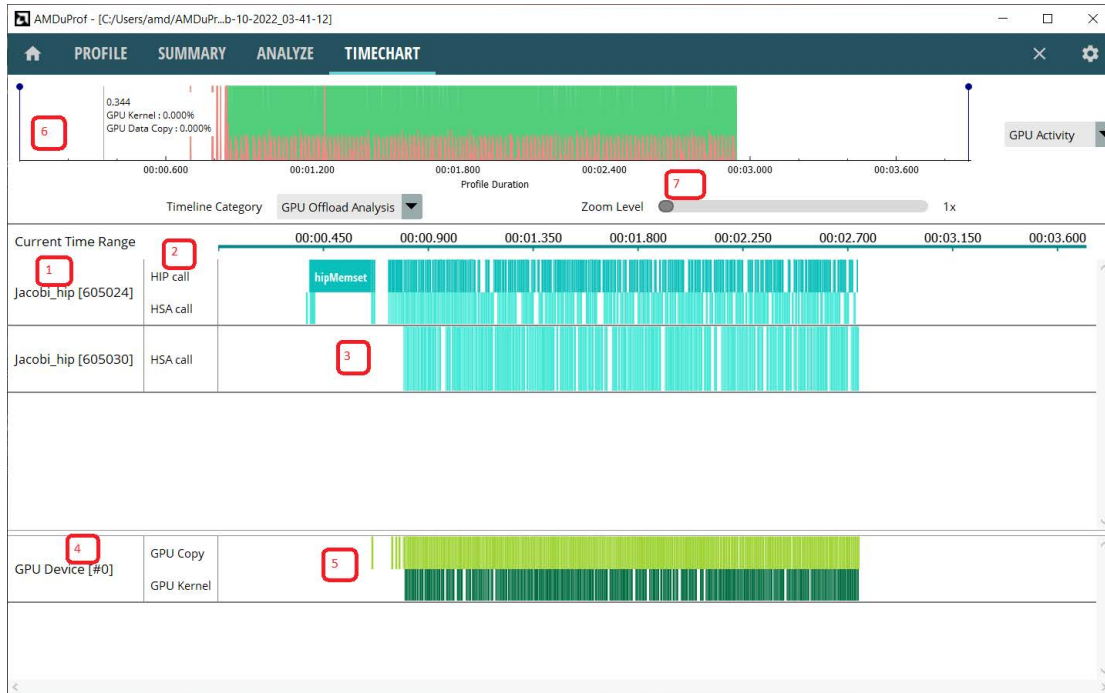


Figure 52. TIMECHART - GPU Offload Analysis

In the above figure:

1. The process or thread name along with the thread ID.
2. The GPU runtime characteristics for which tracing data is available for the current thread.
3. Timeline swim lanes for the various GPU runtime APIs for which tracing data is available.

If a large number of samples are collected for a trace event, then selective samples (selected based on adaptive sampling approach) will be plotted. Either by increasing the zooming using **Zoom Level** slider or by performing selective filtering by selecting interesting time range will load more trace samples for analysis.

4. The GPU device(s) on which the kernels were running.
5. The GPU activities on this GPU device – GPU Copy and GPU Kernel executions.
6. GPU Activity with time range selector – Select the time range so that the GUI plots the trace samples within the selected time range.
7. To zoom in or zoom out, use the **Zoom Level** slider. The zoom in will fetch and render more trace samples.

8.10 GPU Profiling

The AMD ROCprofiler library provides support to monitor GPU hardware performance events when GPU kernels are dispatched and executed. The derived performance metrics are computed and reported in the CSV report. It is supported only with a launch application.

Prerequisites

For GPU performance profiling:

- Requires AMD ROCm 5.2.1 to be installed. For the steps to install AMD ROCm, refer section “Installing ROCm” on page 24

***Note:** Profiling might not work as expected on '5.1.3 or older' and '5.2.3 or later' versions.*

- Supported accelerators - AMD Instinct™ MI100 and MI200

Supported Events and Metrics

The following GPU performance metrics are supported. Run `AMDuProfCLI info --list gpu-events` command to list the supported events on the target system.

The following table shows the list of supported events:

Table 44. Supported Events for GPU Profiling

Event	Description
GRBM_COUNT	GPU free running clock
GRBM_GUI_ACTIVE	GPU busy clock
SQ_WAVES	Count number of waves sent to SQs. (per-simd, emulated, global)
TCC_HIT_sum	Number of cache hits.
TCC_MISS_sum	Number of cache misses. UC reads count as misses.
SQ_INSTS_VALU	Number of VALU instructions issued. (per-simd, emulated)
SQ_INSTS_SALU	Number of SALU instructions issued. (per-simd, emulated)
SQ_INSTS_SMEM	Number of SMEM instructions issued (per-simd, emulated)
SQ_INSTS_LDS	Number of LDS instructions issued (including FLAT) (per-simd, emulated)
SQ_INSTS_GDS	Number of GDS instructions issued (per-simd, emulated)
TCC_EA_RDREQ_sum	Number of TCC/EA read requests (either 32-byte or 64-byte)
TCC_EA_RDREQ_32B_sum	Number of 32-byte TCC/EA read requests
SQ_ACTIVE_INST_VALU	Number of cycles the SQ instruction arbiter is working on a VALU instruction (per-simd, nondeterministic)
SQ_THREAD_CYCLES_VALU	Number of thread-cycles used to execute VALU operations (per-simd)

Table 44. Supported Events for GPU Profiling

Event	Description
TA_FLAT_READ_WAVEFRONTS_sum	Number of flat opcode reads processed by the TA
TA_FLAT_WRITE_WAVEFRONTS_sum	Number of flat opcode writes processed by the TA

The following table shows the list of supported metrics:

Table 45. Supported Metrics for GPU Profiling

Metric	Description
GPU_UTIL (%)	GPU utilization in percentage
VALU_UTIL (%)	VALU utilization in percentage
VALU_THREAD_DIVERGENCE (%)	Average VALU thread divergence in percentage
L2_CACHE_HIT_RATE (%)	Average L2 cache hit rate in percentage
VALU_INSTR (IPW)	Average number of VALU instructions per wave
SALU_INSTR (IPW)	Average number of SALU instructions per wave
SMEM_INSTR (IPW)	Average number of SMEM instructions per wave
LDS_INSTR (IPW)	Average number of LDS instructions per wave
GDS_INSTR (IPW)	Average number of GDS instructions per wave
L2_CACHE_HITS (PW)	Average number of L2 cache hits per wave
L2_CACHE_MISSES (PW)	Average number of L2 cache misses per wave
EA_32B_READ (PW)	Average number of 32-byte reads per wave
EA_64B_READ (PW)	Average number of 64-byte reads per wave
EA_READ_BW (GB/sec)	Read Bandwidth in GB per second

8.10.1 GPU Profiling Using CLI

Collect Profile Data

Use the following command to collect the GPU performance data:

```
$ sudo AMDuProfCLI collect --config gpu -o /tmp/ /home/app/SampleApp
...
Generated data files path: /tmp/AMDuProf-SampleApp-GPUProfile_Dec-09-2021_12-19-27
```

This command will launch the program and collect the profile data. Once the launched application is executed, the AMDuProfCLI will display the session directory path in which the raw profile data are saved.

In the above example, the session directory path is:

```
/tmp/AMDuProf-SampleApp-GPUProfile_Dec-09-2021_12-19-27/
```

Generate Profile Report

Use the following CLI report command to generate the profile report in .csv format by passing the session directory path as the argument to -i option:

```
$ ./AMDuProfCLI report -i /tmp/AMDuProf-SampleApp-GPUProfile_Dec-09-2021_12-19-27
...
Generated report file: /tmp/AMDuProf-SampleApp-GPUProfile_Dec-09-2021_12-19-27/report.csv
```

After processing the data and generating the report, the report file path is displayed on the terminal. An example of the GPU profile report section in the .csv report file is as follows:

GPU PROFILE REPORT										
KERNEL STATS										
Name	Count	Elapsed Time(s)	Avg Elapsed Time(s)	Elapsed Time(s)	EA_READ_	SMEM_INS	VALU_UTIL	L2_CACHE	LDS_INSTR	L2_CACHE
_amd_rocclr_fillBuffer.kd	4501	7.21947	0.00160397	26.9689	13.47	8.42	35.32	85.7	40.69	92.04
void bondedForcesKernel<	501	1.06586	0.00212747	3.98161	4.19	9.43	26.62	88.92	152.42	289.41
void nonbondedForceKern	375	1.59328	0.00424873	5.95181	5.48	31.66	34.85	95.54	692.39	1488.8
void nonbondedForceKern	51	0.236248	0.00463232	0.882524	4.06	47	44.75	96.92	1125.23	1478.37
void modifiedExclusionFor	501	1.43712	0.00286851	5.36848	20.75	27.98	34.41	95.09	625.03	1338.94
void nonbondedForceKern	75	0.513087	0.00684115	1.91668	20.42	33.08	34.19	96.46	940.82	1476.51
void scalar_sum_kernel<flo	126	0.687552	0.00545676	2.56841	25.03	32.24	34.44	96.27	909.84	1424.5
void real_post_process_ke	126	0.280848	0.00222896	1.04913	12.35	8.07	28.92	85.83	110.48	223.82
reduceNonbondedVirialKe	501	2.38511	0.0047607	8.90978	24.41	7.44	38.35	42.89	37.75	56.76
buildBoundingBoxesKerne	51	0.0754965	0.00148032	0.282023	13.16	9.58	26.39	88.56	166.04	325.81
RAW EVENTS										
Name	GRBM_COUNT	GRBM_GUI_AC	SQ_WAVES	TCC_HIT_s	TCC_MISS	SQ_INSTS	SQ_INSTS	SQ_INSTS	SQ_INSTS	SQ_INSTS
_amd_rocclr_fillBuffer.kd	655979072	655940864	9127359	8.4E+08	1.4E+08	3.05E+09	9.57E+08	76825480	3.71E+08	0
void bondedForcesKernel<	334267520	334267520	1257630	3.64E+08	45374256	1.44E+09	4.51E+08	11863202	1.92E+08	0
void nonbondedForceKern	250795680	250795680	1218813	1.81E+09	84609696	1.52E+10	1.67E+09	38583192	8.44E+08	0
void nonbondedForceKern	223318368	223318368	912751	1.35E+09	42918508	1.48E+10	2.85E+09	42899296	1.03E+09	0
void modifiedExclusionFor	220224080	220224080	1102476	1.48E+09	76276864	1.21E+10	1.32E+09	30845012	6.89E+08	0
void nonbondedForceKern	200330336	200330336	1105181	1.63E+09	59872448	1.59E+10	1.49E+09	36560968	1.04E+09	0
void scalar_sum_kernel<flo	167502944	167502944	915902	1.3E+09	50539992	1.27E+10	1.19E+09	29525646	8.33E+08	0
void real_post_process_ke	159802096	159802096	826017	1.85E+08	30512964	7.47E+08	2.44E+08	6665629	91260560	0
reduceNonbondedVirialKe	109840608	109840608	798817	45339272	60364524	3.61E+08	1.32E+08	5944229	30156032	0
buildBoundingBoxesKerne	96050120	96050120	336769	1.1E+08	14174039	4.27E+08	1.32E+08	3224846	55916176	0
DISPATCH STATS										
Name	Avg Grid Size	Max Grid Size	Min Grid Size	Avg Workg	Max Workg	Min Workg	Avg LDS Al	Max LDS A	Min LDS Al	Avg Scratc
void nonbondedForceKern	1145413.02	1165696	1055552	64	64	64	3072	3072	3072	16
void nonbondedForceKern	1033182.72	1165696	938496	64	64	64	2048	2048	2048	16
void nonbondedForceKern	1020388.69	1165696	938496	64	64	64	3072	3072	3072	16
void bondedForcesKernel<	256904.81	365312	220480	64	64	64	512	512	512	0
void modifiedExclusionFor	73920	73920	73920	64	64	64	512	512	512	80
reduceNonbondedVirialKe	96665.8	96768	96256	256	256	256	512	512	512	76
void scalar_sum_kernel<flo	65536	65536	65536	256	256	256	1536	1536	1536	0
void spread_charge_kerne	368896	368896	368896	128	128	128	2048	2048	2048	0

Figure 53. GPU Profile Report

8.11 Other OS Tracing Events

Apart from the OS events that are listed in section “Holistic Analysis” on page 141, following OS events can also be traced along with CPU sampling-based profiles:

Table 46. Supported Events for OS Tracing

Event	Description
pagefault	To trace the number of page faults.
memtrace	To trace memory allocation and deallocation calls. By default, only memory allocations that are \geq 1KB are traced. <i>Note: This is supported only for application level tracing.</i>

Prerequisites

For tracing OS events and runtime libraries:

- Requires Linux kernel 4.7 or later (it is recommended to use kernel 4.15 or later).
- Root access is required to trace the OS events in Linux.
- To install BCC and eBPF scripts, refer section “Installing BCC and eBPF” on page 24. To validate the BCC Installation, run the script `sudo AMDuProfVerifyBpfInstallation.sh`.

8.11.1 Tracing Page Faults and Memory Allocations Using CLI

The AMDuProfCLI can be used to collect the required trace data and generate the report in .csv format for further analysis.

Collect Profile Data

The CLI has an option `--trace` to specify the OS events and runtime libraries to be traced. Example CLI command to trace page faults and memory allocations along with time-based sampling for performing holistic analysis:

```
$ sudo AMDuProfCLI collect --config tbp -trace os=pagefault,memtrace -o /tmp/ /home/app/classic
...
Generated data files path: /tmp/AMDuProf-classic-OSTrace_Dec-09-2021_12-19-27
```

This command will launch the program and collect the profile and trace data. Once the launched application is executed, the AMDuProfCLI will display the session directory path in which the raw profile and trace data are saved.

In the above example, the session directory path is:

`/tmp/AMDuProf-classic-OSTrace_Dec-09-2021_12-19-27/Generate Profile Report`

Use the following CLI report command to generate the profile report in .csv format by passing the session directory path as the argument to `-i` option:

```
$ ./AMDuProfCLI report -i /tmp/AMDuProf-classic-OSTrace_Dec-09-2021_12-19-27
...
Generated report file: /tmp/AMDuProf-classic-OSTrace_Dec-09-2021_12-19-27/report.csv
```

After processing the data and generating the report, the report file path is displayed on the terminal. An example of the GPU trace report section in the .csv report file is as follows:

MONITORED EVENTS					
OS Trace Events:	Name	Threshold	Description		
	PAGEFAULT	0	Page Faults for a process/thread		
	MEMTRACE	1024 bytes	Dynamic Memory Allocation tracing		
OS TRACING REPORT					
PAGEFAULT SUMMARY					
Process	Thread	User PF Count	Kernel PF Count		
/home/amd/SamplePrograms/Scima	"ScimarkStable(196941)"	141	3		
MEMORY ALLOC SUMMARY					
Process	Total Memory Allocated	Total Duration(sec)	Memory Allocated	Memory Deallocation Count	
/home/amd/SamplePrograms/Scima	0.097412	2.37E-05	7	6	

Figure 54. Monitored Events

8.11.2 Tracing Function Call Count using CLI

`funccount` in OS Trace will count the functions of a module (Executable/Library or Kernel Function). The maximum number of functions that can be traced in a single tracing is 1000.

For CLI options, refer to Table 26 on page 93.

An example of the function count report section in the .csv report file is as follows:

FUNCTION COUNT SUMMARY						
Function	Count	Total Time(seconds)	Min Time(seconds)	Max Time(seconds)	Avg Time(seconds)	
main	1	575.689	575.689	575.689	575.689	
kernel_measureMonteCarlo	1	529.388	529.388	529.388	529.388	
MonteCarlo_integrate	1	529.388	529.388	529.388	529.388	
Random_nextDouble	536898960	294.149	5.10E-07	0.000359381	5.48E-07	
kernel_measureLU	1	13.4179	13.4179	13.4179	13.4179	
FUNCTION COUNT DETAIL(From 0 To <5 Sec)						
Function	Process	Thread	Count	Total Time(seconds)	Min Time(seconds)	Max Time(second)
main	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable(118908)		1	575.689	575.689	575.689
FUNCTION COUNT DETAIL(From 5 To <10 Sec)						
Function	Process	Thread	Count	Total Time(seconds)	Min Time(seconds)	Max Time(second)
kernel_measureFFT	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable(118908)		1	8.69741	8.69741	8.69741
RandomVector	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable(118908)		1	0.0020856	0.0020856	0.0020856
FFT_transform	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable(118908)		1	0.000123624	0.000123624	0.000123624
FFT_transform_internal	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable(118908)		1	0.000117512	0.000117512	0.000117512
new_Random_seed	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable(118908)		1	0.000100701	0.000100701	0.000100701
FUNCTION COUNT DETAIL(From 10 To <15 Sec)						
Function	Process	Thread	Count	Total Time(seconds)	Min Time(seconds)	Max Time(second)
FFT_transform_internal	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable(118908)		41	0.00456172	0.000100861	0.000171625
FFT_transform	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable(118908)		24	0.00273375	0.000101231	0.000173209
FFT_inverse	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable(118908)		18	0.00208564	0.000108716	0.00012703

Figure 55. Function Count Summary

Examples:

- Collect the function count of `malloc()` from `libc` called by `AMDTClassicMatMul-bin`; `libc` will be searched for in the default library paths:

```
$ AMDuProfCLI collect --trace os=funccount --func c:malloc -o /tmp/cpuprof-os
AMDTClassicMatMul-bin
```

- Collect context switches, syscalls, pthread API tracing, and function count of malloc() called by AMDTClassicMatMul-bin:

```
$ AMDuProfCLI collect --trace os --func c:malloc -o /tmp/cpuprof-os AMDTClassicMatMul-bin
```

- Collect the count of malloc(), calloc(), and kernel functions that match the pattern 'vfs_read*' system-wide:

```
$ AMDuProfCLI collect --trace os --func c:malloc,calloc,kernel:vfs_read* -o /tmp/cpuprof-os -a -d 10
```

- Collect the count of all the functions from AMDTClassicMatMul-bin:

```
$ AMDuProfCLI collect --trace os=funccount --func /home/amd/AMDTClassicMatMul-bin:--* -o /tmp/cpuprof-os AMDTClassicMatMul-bin
```

8.12 MPI Trace Analysis

MPI trace analysis can be used to analyze, and compute the message passing load imbalance among the ranks of a MPI application running on a cluster. It supports OpenMPI, MPICH, and their derivatives.

The supported thread models are SINGLE, FUNNLED, and SERIALIZED. The profile reports are generated for Point-to-Point and Collective API activity summary.

Support Matrix

- MPI Spec versions: MPI-3.0
- Implementations: OpenMPI, MPICH, and their derivatives
- Languages: C, C++, and Fortran

Tracing Modes

The AMDuProf CLI supports the following 2 modes for MPI tracing:

- LWT – Light-weight tracing is useful for quick analysis of an application. The report gets generated in .csv format on-the-fly during collection stage.
- FULL – Full tracing is useful for in-depth analysis. This mode requires post-processing for report generation in .csv format .

MPI Implementation Support

AMD uProf supports tracing of Open MPI and MPICH and the derivatives:

- trace mpi=mpich for MPICH and derivatives (default option)
- trace mpi=openmpi for Open MPI

Ensure that the correct option (mpich or openmpi) is passed depending on the MPI implementation used for compiling the MPI application. Passing incorrect option might cause undefined behavior.

For more information on MPI tracing options, refer “Linux Specific Options” on page 93.

8.12.1 MPI Light-weight Tracing Using CLI

In LWT mode, quick report gets generated during collection stage. This mode supports limited set of APIs for tracing. This report gives overview of the application runtime activity as follows:

Table 47. List of Supported MPI APIs for Light-weight Tracing

MPI_Bsend	MPI_Recv_init	MPI_Bcast	MPI_Ireduce_scatter
MPI_Bsend_Init	MPI_Rsend	MPI_Gather	MPI_Iscan
MPI_Ibsend	MPI_Rsend_init	MPI_Gatherv	MPI_Iscatter
MPI_Improbe	MPI_Send	MPI_Iallgather	MPI_Iscatterv
MPI_Imrecv	MPI_Send_init	MPI_Iallgatherv	MPI_reduce
MPI_Iprobe	MPI_Ssend	MPI_Iallreduce	MPI_reduce_scatter
MPI_Irecv	MPI_Ssend_Init	MPI_Ialltoall	MPI_Scan
MPI_Irsend	MPI_Allgather	MPI_Ialltoallv	MPI_Scatter
MPI_Isend	MPI_Allgatherv	MPI_Ialltoallw	MPI_Scatterv
MPI_Issend	MPI_Allreduce	MPI_Ibarrier	MPI_Wait
MPI_Mprobe	MPI_Alltoall	MPI_Ibcast	MPI_Waitall
MPI_Mrecv	MPI_Alltoallv	MPI_Igather	MPI_Waitany
MPI_Probe	MPI_Alltoallw	MPI_Igatherv	MPI_Waitsome
MPI_Recv	MPI_Barrier	MPI_Ireduce	

Collect Profile Data

Example of a command to LWT trace an MPI application using AMDuProfCLI:

```
$ mpirun -np <number of processes> ./AMDuProfCLI collect --trace mpi=lwt -o <output_directory>
<application>
```

After completing the tracing, the path to the session directory is displayed on the terminal. LWT report is generated immediately after completing the collection and saved into the session directory in: `<output_directory>/<SESSION_DIR>/mpi/lwt/mpi-summary.csv`.

MPI implementation MPICH or Open MPI should be passed in the command; MPICH is the default.

Following are the sample commands:

```
$ mpirun -np <number of processes> ./AMDuProfCLI collect --trace mpi=lwt,openmpi -o
<output_directory> <application>
```

```
$ mpirun -np <number of processes> ./AMDuProfCLI collect --trace mpi=lwt,mpich -o
<output_directory><application>
```

Ensure that the correct option (mpich or openmpi) is passed depending on the MPI implementation used for compiling the MPI application. Passing an incorrect option might cause undefined behavior.

An example of the LWT report section in the .csv file is as follows:

ENVIRONMENT						
TOTAL RANKS	8					
LIB VERSION	MPICH Version:3.4.2					
STD VERSION	3.1					
MPI FUNCTIONS SUMMARY						
Function	Min Time(seconds)	Max Time(seconds)	Average Time(second)	MPI Time(%)	Volume(Bytes)	Calls
MPI_Probe	0	0.10431	0.01061	0.95905	0	14
MPI_Iprobe	0	0.04699	0	1.776	0	560933
MPI_Wait	0.00003	0.06543	0.01478	3.5307	0	37
MPI_Barrier	0	0.0599	0.04337	17.91578	0	64
MPI_Recv	0	0.00135	0.00013	0.01193	899656	14
MPI_Irecv	0	0.00002	0.00001	0.00162	17046896	48
MPI_Send	0.00001	0.08166	0.02145	1.93841	899656	14
MPI_Isend	0	0.00038	0.00005	0.01567	17046896	48
MPI_Reduce	0.00001	0.00014	0.00005	0.00815	1152	24
MPI_Allreduce	0.00001	1.02037	0.09424	53.52716	3136	88
MPI_Bcast	0.00001	0.10316	0.04426	18.28283	60849496	64
MPI RANK SUMMARY						
Rank	PID	MPI Time(seconds)	MPI Time(%)	Wait Time(seconds)	Call Count	Volume(Bytes)
0	51907	0.77848	5.0246	0.01185	214	13075227
1	51902	2.21421	14.2914	0.07077	247754	11767163
2	51919	1.69282	10.92617	0.06905	196894	12236179
3	51917	2.57916	16.64699	0.11187	139521	11683555
4	51914	1.53032	9.87731	0.08567	106170	12317787
5	51906	2.32437	15.00242	0.04465	178779	11759619
6	51915	1.71808	11.08919	0.04018	36055	12230259
7	51905	2.65584	17.14191	0.113	217701	11677099

Figure 56. LWT Report

8.12.2 MPI Full Tracing Using CLI

Full tracing mode traces more APIs than LWT tracing. This mode is helpful for in-depth analysis of an MPI Application activity.

The report file for the full tracing includes multiple tables to represent various details:

- Communicator summary consists of the following columns:
 - Communicator Id:** Communicator identifier
 - Communicator Size:** Number of the member ranks
 - Elapsed Time:** Time spent by the MPI APIs in the communicator
 - Ranks:** Member rank IDs

- Rank summary consists of the following columns:
 - **Rank:** Rank ID.
 - **PID:** Process ID.
 - **MPI Time (seconds):** Total time spent on the MPI APIs.
 - **MPI Time (%):** Percentage of MPI Time with respect to the total MPI time of all the ranks.
 - **Wait Time (seconds):** Time spent by the rank waiting.
 - **Wait Time (%):** Percentage of the rank wait time with respect to the application runtime.
 - **Call Count:** Number of times MPI APIs are called.
 - **Volume (bytes):** Volume of data in bytes sent or received.
 - **Volume (%):** Percentage of volume with respect to the total volume sent or received by all the ranks.
 - **Elapsed Time (seconds):** Application runtime.
 - **Time (%):** Percentage of elapsed time with respect to the total elapsed time.
- P2P API summary consists of the following columns:
 - **Function:** MPI API name.
 - **Min Time (seconds):** Minimum time of the total time spent in this API in all the ranks.
 - **Max Time (seconds):** Maximum time of the total time spent in this API in all the ranks.
 - **Average Time (seconds):** Average time spent on the API.
 - **MPI Time (%):** Percentage of the time spent on this API with respect to the total time spent on all the MPI APIs.
 - **Volume (Bytes):** Total volume sent or received by this MPI API.
 - **Calls:** Number of times this MPI API is called.
- Communication matrix consists of the following columns:
 - **Rank:** Sender rank ID and receiver rank ID.
 - **MPI Time (seconds):** Total time spent on the APIs sending data from the sender rank to the receiver rank.
 - **MPI Time (%):** Percentage of MPI time with respect to the total MPI Time spent on all the APIs.
 - **Volume (Bytes):** Total volume of data sent from the sender rank to the receiver rank.
 - **Volume (%):** Percentage of volume with respect to the total volume transferred between all the ranks.
 - **Transfers:** Number of transfers from the sender rank to the receiver rank.

- Collective API summary consists of the following columns:
 - **Function:** API name.
 - **Min Time (seconds):** Minimum time spent on this API.
 - **Max Time (seconds):** Maximum time spent on this API.
 - **Average time (seconds):** Average time spent on this API.
 - **MPI Time (%):** Percentage of time spent on this API with respect to the total time spent on all the MPI calls.
 - **Input Volume (Bytes):** Total data in bytes received by all the ranks involved in this API call.
 - **Output Volume (Bytes):** Total data sent by all the ranks involved in this API call.
 - **Calls:** Number of times this API is called.

The list of supported MPI APIs is as follows:

Table 48. MPI APIs

MPI_Pcontrol	MPI_Mrecv	MPI_Reduce	MPI_Iallreduce
MPI_Cancel	MPI_Imrecv	MPI_Allreduce	MPI_Ialltoall
MPI_Probe	MPI_Send	MPI_Alltoall	MPI_Ialltoallv
MPI_Iprobe	MPI_Bsend	MPI_Alltoallv	MPI_Ialltoallw
MPI_Mprobe	MPI_Ssend	MPI_Alltoallw	MPI_Ineighbor_Alltoall
MPI_Improbe	MPI_Rsend	MPI_Neighbor_Alltoall	MPI_Ineighbor_Alltoallw
MPI_Start	MPI_Bsend_init	MPI_Neighbor_Alltoallw	MPI_Ineighbor_Alltoallv
MPI_Startall	MPI_Ssend_init	MPI_Neighbor_Alltoallv	MPI_Ibarrier
MPI_Test	MPI_Rsend_init	MPI_Bcast	MPI_Ibcast
MPI_Testall	MPI_Send_init	MPI_Scan	MPI_Comm_create
MPI_Testany	MPI_Ibsend	MPI_Reduce_Scatter	MPI_Comm_dup
MPI_Testsome	MPI_Issend	MPI_Ireduce_Scatter	MPI_Comm_dup_with_info
MPI_Wait	MPI_Irsend	MPI_Iscan	MPI_Comm_split
MPI_Waitall	MPI_Isend	MPI_Iscatter	MPI_Comm_split_type
MPI_Waitany	MPI_Scatter	MPI_Iscatterv	MPI_Intercomm_create
MPI_Waitsome	MPI_Scatterv	MPI_Igather	MPI_Intercomm_merge
MPI_Barrier	MPI_Gather	MPI_Igatherv	MPI_Cart_create
MPI_Recv	MPI_Gatherv	MPI_Iallgather	MPI_Cart_sub
MPI_Irecv	MPI_Allgather	MPI_Iallgatherv	MPI_Graph_create
MPI_Sendrecv	MPI_Allgatherv	MPI_INeighbor_Allgather	MPI_Dist_graph_create

Table 48. MPI APIs

MPI_Sendrecv_repl ace	MPI_Neighbor_Allgat her	MPI_Ineighbor_Allgat herv	MPI_Dist_graph_create_adjacent
MPI_Recv_Init	MPI_Neighbor_Allgat herv	MPI_Ireduce	

Collect Profile Data

Example of a command to FULL trace an MPI application using AMD uProf CLI:

```
$ mpirun -np <number of processes> ./AMDuProfCLI collect --trace mpi=full -o <output_directory>
<application>
```

After completing the tracing, the path to the session directory is displayed on the terminal.

MPI implementation MPICH or Open MPI should be passed in the command; MPICH is the default.

Following are the sample commands:

```
$ mpirun -np <number of processes> ./AMDuProfCLI collect --trace mpi=full,openmpi -o
<output_directory> <application>
```

```
$ mpirun -np <number of processes> ./AMDuProfCLI collect --trace mpi=full,mpich -o
<output_directory><application>
```

Ensure that the correct option (mpich or openmpi) is passed depending on the MPI implementation used for compiling the MPI application. Passing an incorrect option might cause undefined behavior.

Generate Profile Report

Example of a command to generate the report in .csv format. Pass the session directory path with -i option:

```
$ ./AMDuProfCLI report -i <output_directory>/<SESSION_DIR>
```

After completing the report generation, the *report.csv* file path is displayed on the terminal.

Tables in the Report file

The following screenshots show example sections of a full tracing report file:

MPI TRACING REPORT			
ENVIRONMENT			
Total Ranks	16		
Library version	MPICH Version:3.4.2		
MPI Std Version	3.1		
Thread Model	MPI_THREAD_SINGLE		
MPI COMMUNICATOR SUMMARY (All Ranks)			
Communicator ID	Communicator Size	Elapsed Time(seconds)	Ranks
0	4	0	0;1;2;3;
1	16	613.046	0;1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;
2	4	0	0;4;8;12;
3	4	0	12;13;14;15;
4	4	0	1;5;9;13;
5	4	0	2;6;10;14;
6	4	0	3;7;11;15;
7	4	0	4;5;6;7;
8	4	0	8;9;10;11;

Figure 57. MPI Communicator Summary Table

RANK SUMMARY TABLE										
Rank	PID	MPI Time(seconds)	MPI Time(%)	Wait Time(seconds)	Wait Time(%)	Call Count	Volume(Bytes)	Volume(%)	Elapsed Time(seconds)	Time(%)
0	139011	1.18992	6.82	0	0	154283	12004944	6.25	3.68395	6.26
1	139013	1.23819	7.1	0	0	159097	12004944	6.25	3.68967	6.27
2	139012	1.06928	6.13	0	0	106244	11997024	6.25	3.65075	6.2
3	139014	1.06525	6.11	0	0	130112	11997024	6.25	3.67021	6.23
4	139024	1.21312	6.96	0	0	254896	12004944	6.25	3.69118	6.27
5	139023	1.02078	5.85	0	0	167413	12004944	6.25	3.69623	6.28
6	139031	1.13994	6.54	0	0	228138	11997024	6.25	3.68951	6.27
7	139030	1.07894	6.19	0	0	163684	11997024	6.25	3.69563	6.28
8	139018	1.01404	5.81	0	0	78059	12004944	6.25	3.57768	6.08
9	139017	1.12509	6.45	0	0	190392	12004944	6.25	3.70481	6.29
10	139028	1.25932	7.22	0	0	263437	11997024	6.25	3.67135	6.24
11	139025	0.888971	5.1	0	0	77798	11997024	6.25	3.64134	6.19
12	139037	1.03417	5.93	0	0	177592	12004944	6.25	3.70373	6.29
13	139038	1.05169	6.03	0	0	71849	12004944	6.25	3.71305	6.31
14	139032	1.05608	6.05	0	0	129159	11997024	6.25	3.69288	6.27
15	139036	0.997395	5.72	0	0	142573	11997024	6.25	3.69635	6.28

Figure 58. MPI Rank Summary Table

MPI FUNCTION SUMMARY TABLE (All Ranks)						
Function	Min Time(seconds)	Max Time(seconds)	Average Time(seconds)	MPI Time(%)	Volume(Bytes)	Calls
MPI_Wait	0.00000003	78.6133	0.00332891	88.07	0	1613760
MPI_Irecv	0.00000004	0.00148102	0.000000966	0.01	121926902688.00	806880
MPI_Isend	0.000000061	0.0011091	0.000002503	0.03	103659994656.00	806880
MPI_Comm_dup	0.000016531	0.000523422	0.000206985	0	0	32
MPI_Comm_split	0.000019847	0.000040156	0.000028519	0	0	48
MPI_Cart_create	0.000022733	0.00060768	0.000295566	0	0	32

Figure 59. MPI Point-to-Point API Summary Table

COMMUNICATION MATRIX					
Rank ----> Rank	MPI Time(seconds)	MPI Time(%)	Volume(Bytes)	Volume(%)	Transfers
0 ----> 1	0.0369763	0	1772934048	0.73	16810
0 ----> 4	0.0734582	0	2481247344	1.02	16810
1 ----> 0	0.0289881	0	1772934048	0.73	16810
1 ----> 2	0.0301966	0	1772934048	0.73	16810
1 ----> 5	0.0565452	0	2540520192	1.04	16810
2 ----> 1	0.0343381	0	1772934048	0.73	16810
2 ----> 3	0.0328948	0	1772934048	0.73	16810
2 ----> 6	0.0581532	0	2540520192	1.04	16810
3 ----> 2	0.0353417	0	1772934048	0.73	16810
3 ----> 7	0.0633404	0	2504096160	1.03	16810
4 ----> 0	0.0363977	0	2481247344	1.02	16810
4 ----> 5	0.0274887	0	1832206896	0.75	16810
4 ----> 8	0.0665892	0	2481247344	1.02	16810
5 ----> 1	0.0326747	0	2540520192	1.04	16810
5 ----> 4	0.0225724	0	1832206896	0.75	16810
5 ----> 6	0.029573	0	1832206896	0.75	16810
5 ----> 9	0.0645192	0	2540520192	1.04	16810
6 ----> 2	0.0428896	0	2540520192	1.04	16810
6 ----> 5	0.0323074	0	1832206896	0.75	16810
6 ----> 7	0.0325295	0	1832206896	0.75	16810

Figure 60. MPI Communication Matrix

COLLECTIVE EVENTS SUMMARY							
Function	Min Time(seconds)	Max Time(sec)	Average Time(se	MPI Time(%)	Input Volume(Byte	Output Volume(Bytes	Calls
MPI_Scatterv	0.000000049	0.110154	0.000033863	0.32	493951444	493951444	573760
MPI_Gather	0.000000016	3.67246	0.000763753	0.73	233984	233984	58496
MPI_Gatherv	0.000000014	1.72094	0.00146369	1.17	5855507368	5855507368	48704
MPI_Allreduce	0.000004759	0.0151899	0.00302453	0	448	448	80
MPI_Bcast	0.000000231	5.48656	0.00504196	9.66	5125009020	341667268	116880

Figure 61. MPI Collective API Summary Table

8.12.3 MPI FULL Tracing Using GUI

Collecting and Importing a Trace

Use CLI to trace a target MPI application and generate the report using CLI. For the steps, refer section “MPI Full Tracing Using CLI” on page 162. Import the report to GUI as shown in the following figure to analyze the trace data:

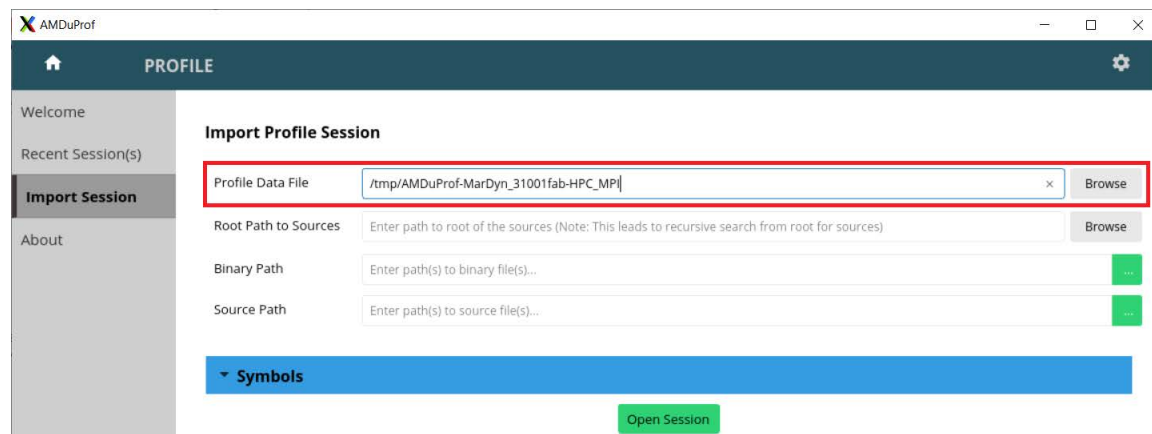


Figure 62. Import Profile Session

Analyzing MPI Communication Matrix

After the import is complete, use **MPI Communication Matrix** view to analyze the MPI trace data in the GUI. Navigate to **HPC > MPI Flat Profile** to view the MPI communication matrix visualizer. This view displays rank-to-rank communication summary in matrix format. The x and y-axis in the matrix are receiver and sender ranks respectively.

Following figure shows the MPI communication matrix:

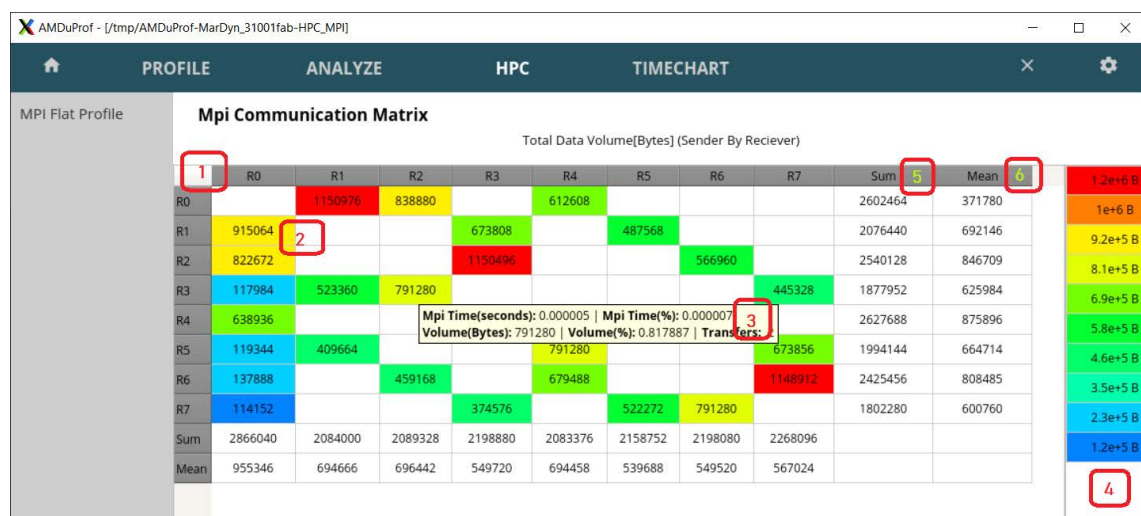


Figure 63. MPI Communication Matrix

In the above figure:

1. Ranks ordered in row-wise and column-wise.
2. Each cell displays the total data volume transferred from one rank to another rank.
3. Tool-tip shows additional details when the mouse is hovered over a cell.
4. Color-coding legend based on data volume.
5. Sum of all the data transfers for the rank.
6. Mean of all the data transfers for the rank.

Analyzing MPI Rank Analysis View

Navigate to **TIMECHART > MPI Rank Analysis** in the **Timeline Category** drop-down. This view displays MPI API calling activities in the timeline graph as follows:

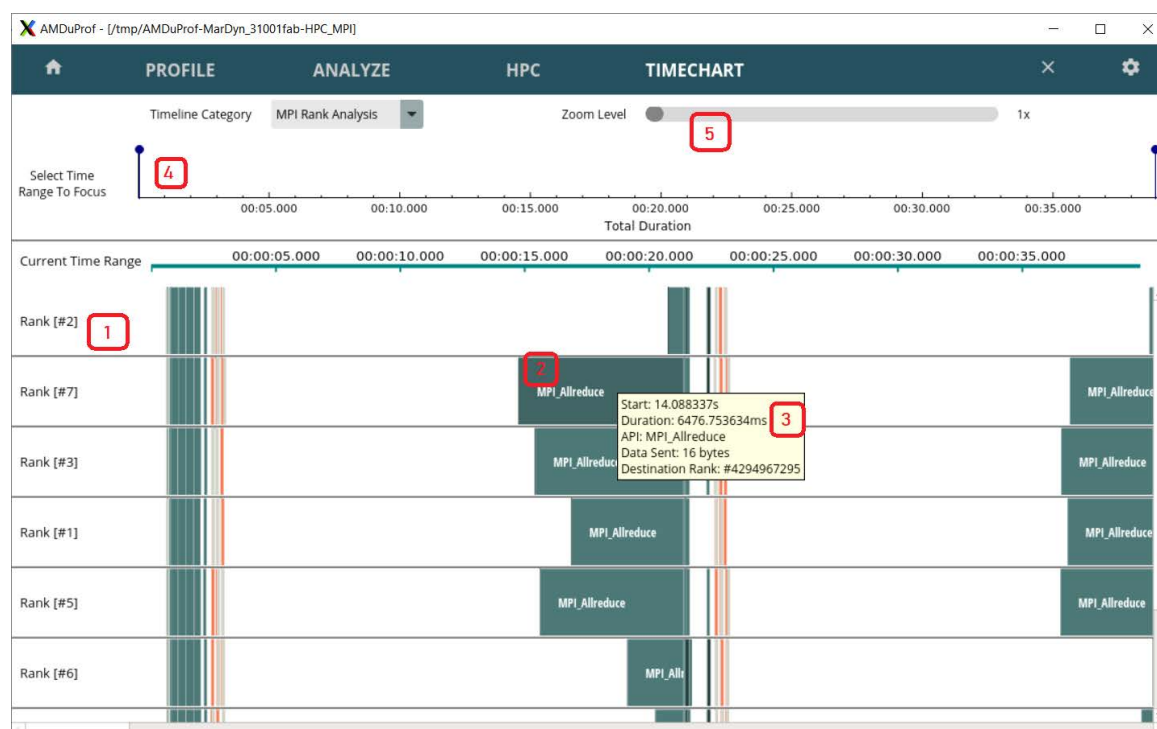


Figure 64. TIMECHART - MPI Rank Analysis

In the above screenshot:

1. Rank ID
2. MPI API activity
3. Tool-tip shows more info about the MPI API activity
4. To select a specific time-range
5. To zoom in/out the visualizer

Chapter 9 Power Profile

9.1 Overview

System-wide Power Profile

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

These counters are collected from various resources such as RAPL and MSRs. They are collected at regular time interval and either reported as a text file or plotted as line graphs. They can also be saved into the database for future analysis.

Features

AMD uProf comprises of the following features:

- The GUI can be used to configure and monitor the supported power metrics.
- The TIMECHART page helps to monitor and analyze:
 - Logical Core level metrics – Core Effective Frequency and P-State
 - Physical Core level metrics – RAPL based Core Power and Temperature
 - Package level metrics – RAPL based Package Power
- AMDuProfCLI timechart command collects the system metrics and writes into a text file or comma-separated-value (CSV) file.
- API library allows you to configure and collect the supported system level performance, thermal and power metrics of AMD CPU/APUs.
- The collected live profile data can be stored in the database for future analysis.

9.2 Metrics

The supported metrics depend on the processor family and model and are broadly grouped under various categories. Following are the supported counter categories by processor families:

Table 49. Family 17h Model 00h – 0Fh (AMD Ryzen™, AMD Ryzen ThreadRipper™, and 1st Gen AMD EPYC™)

Power Counter Category	Description
Power	Average Power for the sampling period, reported in Watts. This is an estimated consumption value based on the platform activity levels. It is available for Core and Package.
Frequency	CPU Core Effective Frequency for the sampling period, reported in MHz.

Table 49. Family 17h Model 00h – 0Fh (AMD Ryzen™, AMD Ryzen ThreadRipper™, and 1st Gen AMD EPYC™)

Power Counter Category	Description
Temperature	Average temperature for the sampling period, reported in Celsius. The temperature reported is with reference to Tctl. It is available for Package.
P-State	CPU P-State at the time when sampling was performed.

Table 50. Family 17h Model 10h – 1Fh (AMD Ryzen™ and AMD 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	CPU 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 P-State at the time when sampling was performed.

Table 51. Family 17h Model 70h – 7Fh (3rd Gen AMD 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	CPU Core Effective Frequency for the sampling period, reported in MHz
P-State	CPU 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.

Table 52. 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	CPU Core Effective Frequency for the sampling period, reported in MHz
P-State	CPU P-State at the time when sampling was performed.

Table 52. Family 17h Model 30h – 3Fh (EPYC 7002)

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

Table 53. Family 19h Model 0h – 2Fh (EPYC 7003 and EPYC 9000)

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	CPU Core Effective Frequency for the sampling period, reported in MHz
P-State	CPU 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.

9.3 Using Profile through 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 database. Complete the following steps to configure and start the profile:

9.3.1 Configuring a Profile

Complete the following steps to configure a profile:

1. Click the **PROFILE** tab at the top navigation bar or one of the following on the *Welcome* page:
 - **Profile entire System**
 - **See What's guzzling power in your system**
 The *Select Profile Target* page is displayed.
2. Click the **Next** button.
The *Select Profile Type* page is displayed.

- From the **Select Profile Type** drop-down, select the option **System-wide Power Profile (Live)**.

All the live profiling options and available counters are displayed in the respective panes as follows:

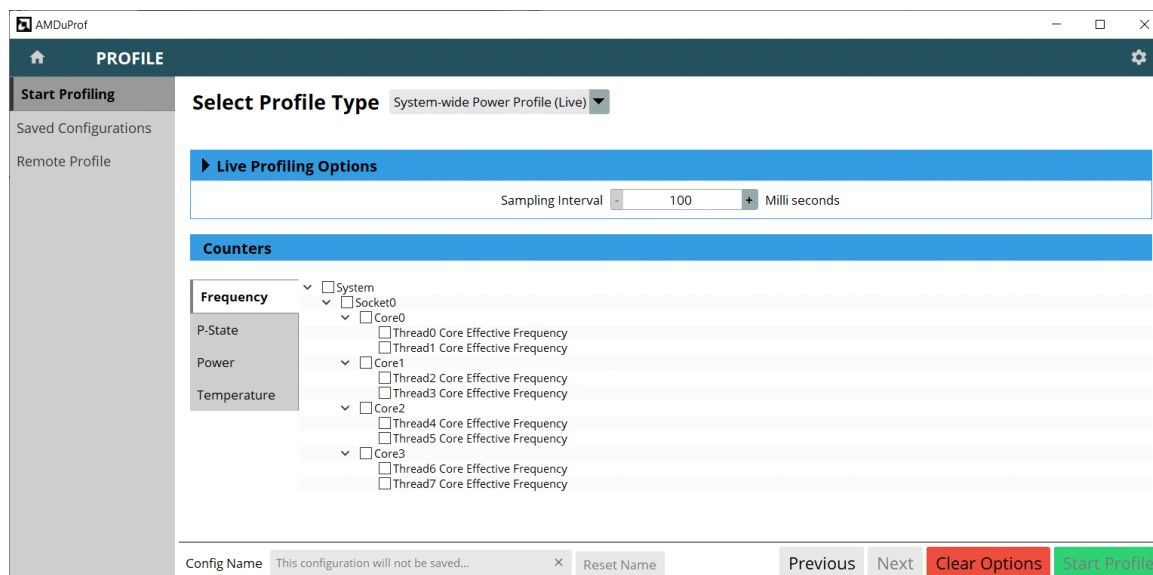


Figure 65. Live System-wide Power Profile

- In the *Counters* pane, select the required counter category and the respective options.

Note: You can configure multiple counter categories.

During the profiling, you can render the graphs live.

- Click the **Start Profile** button.

In this profile type, the profile data will be generated as line graphs in the *TIMECHART* page for further analysis.

9.3.2 Analyzing a Profile

Once the required counters are selected and the profile data collection begins, the **TIMECHART** tab will open and the metrics will be plotted in the live timeline graphs.



Figure 66. Timechart Page

1. In the *TIMECHART* page, the metrics will be plotted in the live timeline graphs. The line graphs are grouped together and plotted based on the category.
2. There is a data table adjacent to each graph to display the current value of the counters.
3. From the *Graph Visibility* pane, you can choose the graph to display.
4. When plotting is in progress, you can:
 - Click the **Pause Graphs** button to pause the graphs without pausing the data collection. You can click the **Play Graphs** button to resume them later.
 - Click the **Stop Profiling** button to stop the profiling without closing the view. This will stop collecting the profile data.
 - Click the **Close View** button to stop the profiling and close the view.

9.4 Using CLI to Profile

You can use AMDuProfCLI timechart command to collect the system metrics and write them into a text file or comma-separated-value (CSV) file. To collect power profile counter values, complete the following steps:

1. Run the command with `--list` option to get the list of supported counter categories.
2. Use the command to specify the required counters with `-e` or `--event` option to collect and report the required counters.

The timechart run to list the supported counter categories is as follows:

```
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>
```

Figure 67. --list Command Output

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

```
C:\Program Files\AMD\AMDuProf\bin>AMDuProfCLI.exe timechart -e Power,Frequency -o c:\Temp\power-prof "c:\Program Files\AMD\AMDuProf\
examples\AMDTClassicMatMul\bin\AMDTClassicMatMul.exe"
```

Profiling started...

Matrix multiplication sample
=====

Initializing matrices
Multiplying matrices

Invoke inefficient implementation of matrix multiplication
Elapsed time: 1.6530 sec (0.0010 sec resolution)

Profile finished
Generated data files path: c:\Temp\power-prof\AMDuProf-AMDTClassicMatMul-Timechart_Dec-20-2021_16-20-54
Live Profile Output file : c:\Temp\power-prof\AMDuProf-AMDTClassicMatMul-Timechart_Dec-20-2021_16-20-54\timechart.csv

```
C:\Program Files\AMD\AMDuProf\bin>
```

Figure 68. Timechart Run

The above run will collect the power 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.

9.4.1 Examples

Windows

- Collect all the power counter values for a duration of 10 seconds with a 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 adding the results to a csv file:

```
C:\> AMDuProfCLI.exe timechart --event frequency -o C:\Temp\Poweroutput --interval 500 --duration 10
```

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

```
C:\> AMDuProfCLI.exe timechart --event core=0-3,frequency -o C:\Temp\Poweroutput --interval 500 --duration 10 --format txt
```

Linux

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

```
$ ./AMDuProfCLI timechart --event power --interval 100 --duration 10
```

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

```
$ ./AMDuProfCLI timechart --event frequency -o /tmp/PowerOutput --interval 500 --duration 10
```

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

```
$ ./AMDuProfCLI timechart --event core=0-3,frequency -o /tmp/PowerOutput --interval 500 --duration 10 --format txt
```

9.5 AMDPowerProfileAPI Library

API library allow you to configure and collect the supported power profiling counters on various AMD platforms directly without using AMD uProf GUI or CLI. The AMDPowerProfileAPI library is used to analyze the power efficiency of systems based on AMD CPUs and APUs.

These APIs provide interface to read the power, thermal, and frequency characteristics of AMD CPUs and APUs 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(s).

For a detailed information on these APIs, refer AMDPowerProfilerAPI.pdf in the AMD uProf installation folder.

9.5.1 Using the APIs

Refer the sample program *CollectAllCounters.cpp* on how to use these APIs. The program must be linked with the 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 the directory `<AMDuProf-install-dir>/Examples/CollectAllCounters/`. To build and execute the sample application, complete the following steps based on the OS that you are using:

Windows

A Visual Studio 2015 solution file *CollectAllCounters.sln* is available at the directory *C:/Program Files/AMD/AMDuProf/Examples/CollectAllCounters/* to build the sample program.

Linux

1. Execute the following commands 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>/lib -Wl,-rpath <AMDuProf-install-dir>/bin -o
CollectAllCounters
```

2. Run the following commands to execute:

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

9.6 Limitations

- Only one power profile session can run at a time.
- Minimum supported sampling period in CLI is 100ms. It is recommended to use a large sampling period to reduce the sampling and rendering overhead.

Chapter 10 Energy Analysis

10.1 Overview

AMD uProf profiler offers Energy 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.

Note: This feature is not supported on AMD “Zen4” processors.

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, and 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, or Source line
 - To correlate the profile data to Function and Source line, the debug information generated 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
- Hot spots summary
- Process and function analysis
- Source and disassembly analysis

10.2 Using CLI to Profile

To profile and analyze the performance of a native (C/C++) application, complete the following steps:

1. Prepare the application. For more information, refer “Reference” on page 193.
2. Collect the samples for the application using AMDuProfCLI collect command.
3. Generate the report using AMDuProfCLI report command in a 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), collect the profile data, and 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. Then it writes them into a database and generates a report in CSV format.

Example

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

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

- Generate a report from the session data folder:

```
C:\> AMDuProfCLI.exe report -i c:\Temp\pwrprof\<session-dir>
```

- Generate a 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\<session-dir>
```

10.3 Limitations

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

Chapter 11 Remote Profiling

11.1 Overview

AMD uProf has the ability to connect to remote systems and trigger collection, translation of data on the remote system and then visualize it in local GUI.

Note: CLI does not support remote profiling.

AMD uProf uses a separate AMDProfilerService binary that can be launched as an application server on the remote target and local GUI can connect to such a server. By default, authorization must be set up on the server to connect to the local GUI. Complete the following steps:

1. Locate the local GUI client ID.
2. Authorize the client ID on the remote target to connect to AMDProfilerService.
3. Launch AMDProfilerService with appropriate options/permissions on remote target.
4. Specify the connection details in the local GUI to connect to the remote target.
5. Local GUI updates itself and displays the remote data (including settings, session history, available events for profiling/tracing, and so on).
6. Proceed to import session/profile on the remote target.
7. When you are done with remote target, disconnect to update the local data in GUI.

Support

Remote profiling from Windows (host/local platform) to Linux (target/remote platform) is supported.

11.2 Setting up Authorization

Complete the following steps to set up the authorization:

1. Navigating to **PROFILE > Remote Profile** and locate **Client ID**:

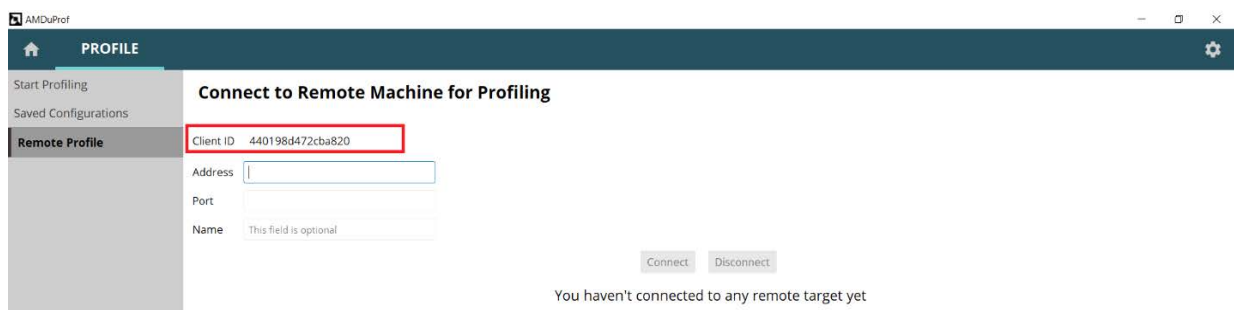


Figure 69. Client ID

2. Copy the **Client ID** (alphanumeric value).
3. On remote target, navigate to the AMD uProf bin directory and execute the following command:

```
AMDProfilerService --add <client_id>
```

This will authorize the client to connect to this remote target.

To revoke the authorization, execute the following command:

```
AMDProfilerService --clear-user <client_id>
```

11.3 Launching AMDProfilerService

Specify the binding IP address to launch AMDProfilerService as an application server:

```
AMDProfilerService --ip 127.0.0.1
```

This IP address should be one of the IP addresses of the target/remote machine on which AMDProfilerService is launched.

If target/remote machine has multiple IP addresses, the `ping` command can be used on the host/local machine to determine which IP address (of the remote machine) is reachable from the local machine. The reachable IP address can be passed to `--ip` option.

(Optional) You can specify the following options:

Table 54. AMDProfilerService Options

Option	Description
<code>--port <port_number></code>	Specify the port number
<code>--logpath <path></code>	Specify the log file path
<code>--bypass-auth</code>	Skip the authorization <i>Note: This option must be used with caution as it will skip the authorization.</i>
<code>--fsearch-depth <depth></code>	Specify the maximum depth for recursive file search operations <i>Note: This option is applicable only for importing a session from the GUI.</i>
<code>--fsearch-timeout <timeout></code>	Specify the maximum duration (in seconds) for recursive file search operations <i>Note: This option is applicable only for importing a session from the GUI.</i>

Following is the sample screen of remote profiling connection establishment:

```
$ ./AMDProfilerService --ip 10.138.152.101 --port 32768
AMDuProf service started...
Listening for connection on port 32768 ...
```

Figure 70. Remote Profiling Connection Establishment

Following is the sample screen of IP selection:

```
-bash-4.4$ ./AMDProfilerService
IP address not specified, found IP address(es) to bind.
Select any one (Type the option number and press return)

  1. 127.0.0.1      (Adapter: lo)
  2. 10.138.136.239 (Adapter: enp97s0)
  3. 10.138.139.51  (Adapter: ens4)
  4. 192.168.122.1  (Adapter: virbr0)

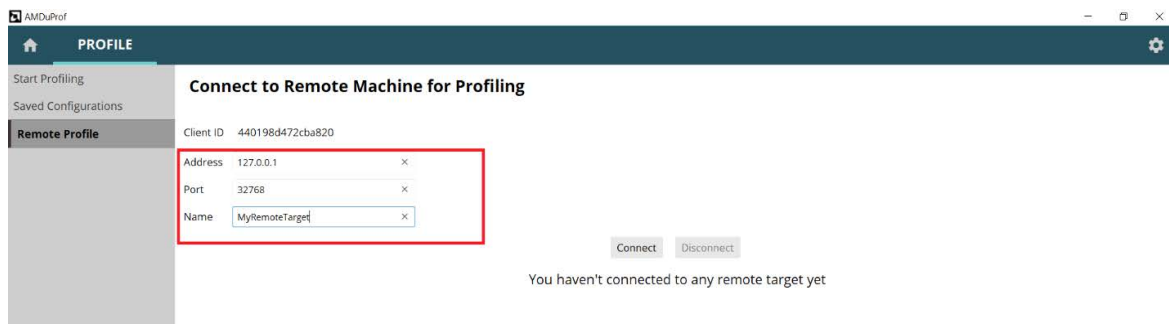
Specify option (1-4): █
```

Figure 71. Selecting IP

11.4 Connecting to Remote Target

Complete the following steps to connect the remote target:

1. Once AMDProfilerService is launched on the remote target, go to the **Remote Profile** page and specify the IP address, port number, and optional name for the remote target as follows:



The screenshot shows the AMDuProf application window with the 'Remote Profile' tab selected. The main area is titled 'Connect to Remote Machine for Profiling'. It displays a 'Client ID' of 440198d472cba820. Below this, there are three input fields: 'Address' (127.0.0.1), 'Port' (32768), and 'Name' (MyRemoteTarget). These fields are enclosed in a red rectangular box. To the right of the fields are 'Connect' and 'Disconnect' buttons. At the bottom, a message states 'You haven't connected to any remote target yet'.

Figure 72. Connect to Remote Machine

2. Click the **Connect** button.

The remote target data is displayed after a few seconds. All the profiling steps or importing session steps remain identical as local henceforth. Once connected, the provided IP, port, and name are saved as follows:

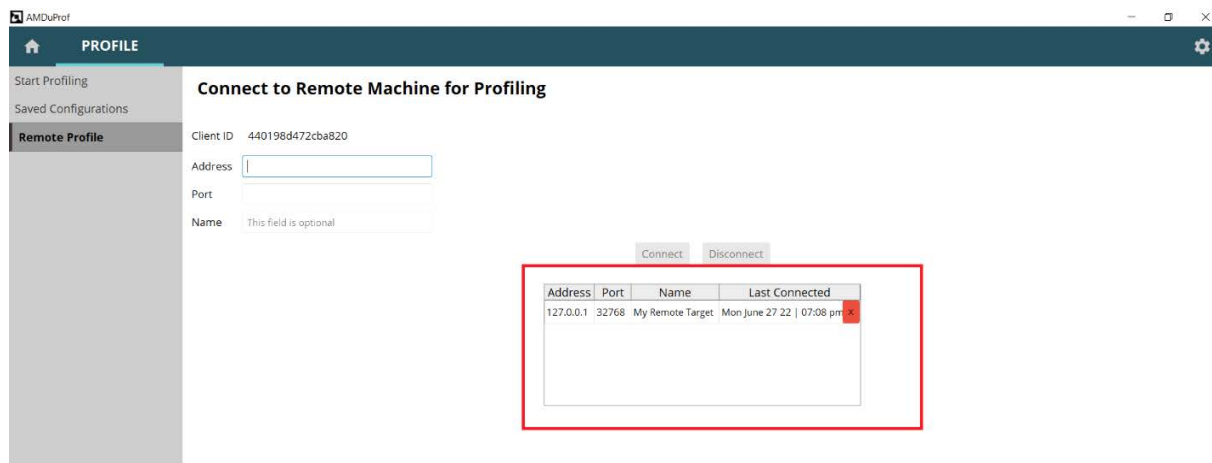


Figure 73. Remote Target Data

You can double-click on any table entry containing IP address to load the corresponding details and connect to the required remote target.

Once connected, the title bar will reflect the connection to the remote target, **Disconnect** button in the **Remote Profile** page will be enabled (instead of the **Connect** button) as follows:

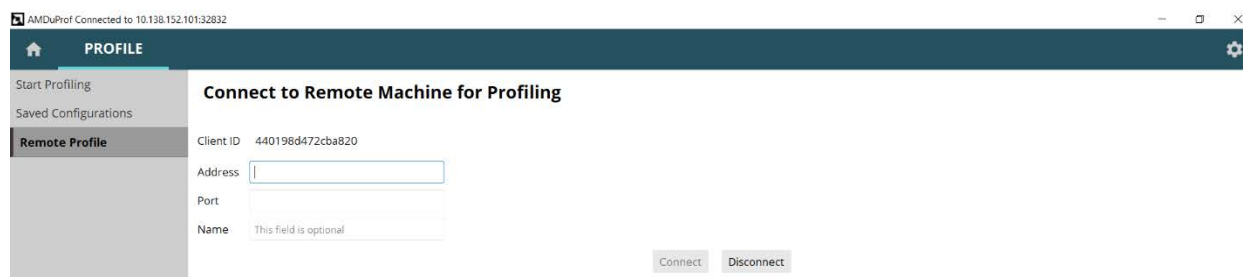


Figure 74. Disconnect Button

11.5 Limitations

- Once connected to a remote target, all the **Browse** buttons in the GUI will remain disabled. You can copy/paste or type the URI paths wherever required.
- If you have not closed the GUI after profiling locally and try to connect to **Remote Target**, the GUI may crash sometimes. Hence, it is recommended to close the GUI after local profiling if remote connection is desired.

- If local data is not required and you try to connect to the same remote target frequently, use the following command to directly connect to the remote target (if it is running):

```
AMDuProf <ip_address> <port>
```

For example, AMDuProf 127.0.0.1 32768

- A client (GUI instance) can connect to a AMDProfilerService instance. However, if multiple instances of the GUI are launched by a user, only one will succeed. Different users can connect to the same AMDProfilerService as they will have different client IDs.
- Multiple instances of AMDProfilerService can be launched. However, all of them must be on different ports even if they are bound to the same IP address.
- Remote profiling connection establishment might fail if the target system firewall is enabled. In such cases, disable the firewall or add an exception for AMDProfilerService in the firewall rules of the target system and try reconnecting. Another reason for failure could be unavailability of port number. This can happen due to network configuration, firewall settings, or another program blocking usable ports.
- Profiling of MPI applications is not supported with remote profiling.

Chapter 12 AMD uProf Virtualization Support

12.1 Overview

AMD uProf supports profiling in the virtualized environments. Availability of the profiling features depends on the counters virtualized by the hypervisor manager. Currently, AMD uProf supports the following hypervisors (with Linux and Windows OS as guest on these virtualized environments):

- VMWare ESXi
- Microsoft Hyper-V
- Linux KVM
- Citrix Xen

Feature support matrix on various hypervisors:

Table 55. AMD uProf Virtualization Support

Features	Microsoft Hyper-V			KVM		VMware ESXi		Citrix Xen	
	Host Root Partition (system mode)	Host Root Partition	Guest VMs	Host	Guest VMs	Host	Guest VMs	Host	Guest VMs
CPU Profiling									
Timer Based Profiling (TBP)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Micro-architecture Analysis (EBP)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
Instruction Based Sampling (IBS)	Yes	No	No	No	No	No	No	No	No
Cache Analysis	Yes	No	No	No	No	No	No	No	No
HPC – MPI Code Profiling	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HPC – OpenMP Tracing	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HPC – MPI Tracing	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
OS Tracing	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Power Profiling									

Table 55. AMD uProf Virtualization Support

Features	Microsoft Hyper-V			KVM		VMware ESXi		Citrix Xen	
	Host Root Partition (system mode)	Host Root Partition	Guest VMs	Host	Guest VMs	Host	Guest VMs	Host	Guest VMs
Live Power Profile	No	No	No	No	No	No	No	No	No
Power Application Analysis	No	No	No	No	No	No	No	No	No
User Interface									
Graphical Interface	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Command Line	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
API									
Profile Control API	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Power Profiler API	No	No	No	No	No	No	No	No	No
System Analysis									
AMDuProfPCM	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
AMDuProfSys	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No

Note: The virtualized hardware counters need to be enabled while configuring the guest VMs on the respective hypervisors.

12.2 CPU Profiling

CPU Profiling supports:

- Profiling of guest VM from guest VM.
- Profiling of guest VM from host system (KVM hypervisor).

12.2.1 Profiling of Guest VM from Guest VM

Timer based profiling can be performed on all the supported Host and Guest VMs, whereas the hardware counter profiling is completely dependent on the vPMUs exposed by the hypervisor.

12.2.2 Profiling of Guest VM from Host System (KVM Hypervisor)

This feature supports profiling of KVM guest OS kernel and kernel modules (*.ko) from the host. The following features are supported:

- Collection of PMU samples on guest OS
- Profiling of guest OS and/or host OS
- System wide profiling to profile KVM-guest and other running processes

The following features are not supported:

- Call stack
- Attach to process
- Launch application

12.2.3 Preparing Host system to Profile Guest Kernel Modules

Before beginning the profiling on the guest OS, the following files must be copied on the host machine to facilitate symbol resolution for the guest VMs:

1. Copy `/proc/kallsyms` and `/proc/modules` from the guest OS to the host machine.
2. Copy guest `vmlinux` and kernel sources in a folder on a host system.

These files should belong to the guest VM whose PID is provided as an argument to `--guest-kvm` option.

12.2.4 AMD uProf CLI with Profiling Options

AMD uProf CLI contains the following options to support the guest OS profiling from the host OS:

```
$ ./AMDuProfCLI collect [--kvm-guest <pid>] [--guest-kallsyms <path>] [--guest-modules <path>]
[--guest-search-path <path>] ....
```

The following table lists various `collect` command options:

Table 56. AMD uProf CLI Collect Command Options

Arguments	Options	Description
<code>--kvm-guest</code>	PID of <code>qemu-kvm</code> process to be profiled	Collect guest-side performance profile. This option collects KVM guest symbols information.
<code>--guest-kallsyms</code>	Path of <code>guest /proc/kallsyms</code> copied on local host	Guest OS <code>/proc/kallsyms</code> file copy. AMD uProf reads it to get guest kernel symbols. You can copy it from the guest OS.

Table 56. AMD uProf CLI Collect Command Options

Arguments	Options	Description
--guest-modules	Path of <i>guest /proc/modules</i> copied on local host	Guest OS <i>/proc/modules</i> file copy. AMD uProf reads it to get the guest kernel module information. You can copy it from the guest OS.
--guest-search-path	Path of guest vmlinux and kernel sources copied on local host	Guest OS vmlinux and search directory. AMD uProf reads it to resolve the guest kernel module information. You can copy it from the guest OS.

12.2.5 Examples

- Get the kvm guest OS PID:

```
$ ps aux | grep kvm
```

- Collecting pmcx76 event data for 10 secs (for guest kallsyms and guest kernel modules)

```
$ ./AMDuProfCLI collect -e event=pmcx76,interval=250000 -o /tmp/cpuprof-76-guest-only -d 10 -  
-kvm-guest 2444 --guest-kallsyms /home/amd/guest/guest-kallsyms --guest-modules /home/amd/  
guest/guest-module
```

Generate report from the collected data:

```
$ ./AMDuProfCLI report -i /tmp/cpuprof-76-guest-only/AMDuProf-SWP-EBP_Nov-08-2021_15-00-33
```

- Collecting pmcx76 event data for 10 secs (for guest kallsyms):

```
$ ./AMDuProfCLI collect -e event=pmcx76,interval=250000 -o /tmp/cpuprof-76-guest-only -d 10 -  
-kvm-guest 2444 --guest-kallsyms /home/amd/guest/guest-kallsyms
```

Generate report from the collected data:

```
$ ./AMDuProfCLI report -i /tmp/cpuprof-76-guest-only/AMDuProf-SWP-EBP_Nov-08-2021_15-00-33
```

- Collecting system-wide samples for pmcx76 event data for 10 secs (for guest kallsyms and guest kernel modules):

```
$ ./AMDuProfCLI collect -e event=pmcx76,interval=250000 -o /tmp/cpuprof-76-guest-only -d 10 -  
-kvm-guest 2444 --guest-kallsyms /home/amd/guest/guest-kallsyms --guest-modules /home/amd/  
guest/guest-module -a
```

Generate report from the collected data:

```
$ ./AMDuProfCLI report -i /tmp/cpuprof-76-guest-only/AMDuProf-SWP-EBP_Nov-08-2021_15-00-33
```

- Collecting system-wide samples for pmcx76 event data for 10 secs (for guest kallsyms):

```
$ ./AMDuProfCLI collect -e event=pmcx76,interval=250000 -o /tmp/cpuprof-76-guest-only -d 10 -  
-kvm-guest 2444 --guest-kallsyms /home/amd/guest/guest-kallsyms -a
```

Generate report from the collected data

```
$ ./AMDuProfCLI report -i /tmp/cpuprof-76-guest-only/AMDuProf-SWP-EBP_Nov-08-2021_15-00-33
```

12.3 AMDuProfPcm

AMDuProfPcm is based on the following hardware and OS primitives provided by host or guest operating system. Run the command `./AMDuProfCLI info --system` to obtain this information and look for the following sections:

```
[PERF Features Availability]
  C ore PMC      : Yes (Requires to collect dc, fp, ipc, l1, l2 metrics)
  L3 PMC         : Yes (Requires to collect l3 metrics option)
  DF PMC         : Yes (Requires to collect memory, xgmi, pcie metrics)
  PERF TS       : No

[RAPL/CEF Features Availability]
  RAPL           : Yes
  APERF & MPERF : Yes (Requires to collect cpu "Utilization" and Effective
Frequency)
  Read Only APERF & MPERF: Yes (Requires to collect cpu "Utilization" and Effective
Frequency)
  IRPERF         : Yes
  HW P-State Control : Yes
```

In Linux environment, check if the `msr` module is available and can be loaded using following command:

```
$ modprobe msr
```

12.4 AMDuProfSys

AMDuProfSys is based on the following hardware and OS primitives provided by host or guest operating system. Run the command `./AMDuProfCLI info --system` to obtain this information and look for the following sections:

```
[PERF Features Availability]
  Core PMC      : Yes (Requires to collect core metrics)
  L3 PMC        : Yes (Requires to collect l3 metrics)
  DF PMC        : Yes (Requires to collect df metrics)
  PERF TS       : No

[RAPL/CEF Features Availability]
  RAPL           : Yes
  APERF & MPERF : Yes (Requires to collect cpu "Utilization" and Effective
Frequency)
  Read Only APERF & MPERF: Yes (Requires to collect cpu "Utilization" and Effective
Frequency)
  IRPERF         : Yes
  HW P-State Control : Yes
```

In Linux environment, check if Linux kernel `perf` module and user space tools are available.

Chapter 13 Profile Control APIs

13.1 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, that is, 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, for example, a CPU intensive loop and a hot function.

The target application needs to be recompiled after instrumenting the application to enable/disable profiling of the required code regions only.

Header Files

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

Static Library

The instrumented application should link with the AMDProfileController static library available in:

Windows

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

Linux

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

13.1.1 CPU Profile Control APIs

These profile control APIs are available to pause and resume the CPU profile data collection in a C or C++ application.

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_PROFILE_CPU);
```

amdProfilePause

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

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

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

13.1.2 Using 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 the CPU Profiler.

These APIs can be:

- Called multiple times to profile different parts of the code.
- Spread across multiple functions, that is, resume called from one function and stop called from another function.
- Spread across threads, that is, resume called from one thread and stop called from another thread of the same target application.

In the following 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;
}
```

13.1.3 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 it 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 the *-static* option while compiling with g++.

13.1.4 Profiling Instrumented Target Application

AMD uProf GUI

After compiling the target application, create a profile configuration in AMD uProf, 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 are done, start the CPU profiling. The profiling will begin in the paused state and the target application execution begins. When the resume API is called from target application, CPU Profile starts profiling till pause API is called from the target application or the application is terminated. When the pause API is called in the target application, the profiler stops profiling and waits for the next control API call.

AMDuProfCLI

To profile from CLI, option *--start-paused* should be used to start the profiler in a paused 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/cpumprof-tbp /tmp/AMDuProf/  
Examples/ClassicCpuProfileCtrl/ClassicCpuProfileCtrl
```

13.1.5 Limitations

The CPU profile control APIs are not supported for the MPI applications.

Chapter 14 Reference

14.1 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.

14.1.1 Generating 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, complete the following steps:

1. Right-click the project and select **Properties** from the menu.
2. From the **Configuration** drop-down, select **Active(Release)**.
3. From the **Platform** drop-down, select **Active(Win32)** or **Active(x64)**.
4. In the project pane on the left, expand **Configuration Properties**.
5. Expand **C/C++** and select **General**.
6. In the work pane, select **Debug Information Format**.

7. From the drop-down, select **Program Database (/ZI)** or **Program Database for Edit & Continue (/ZL)**.

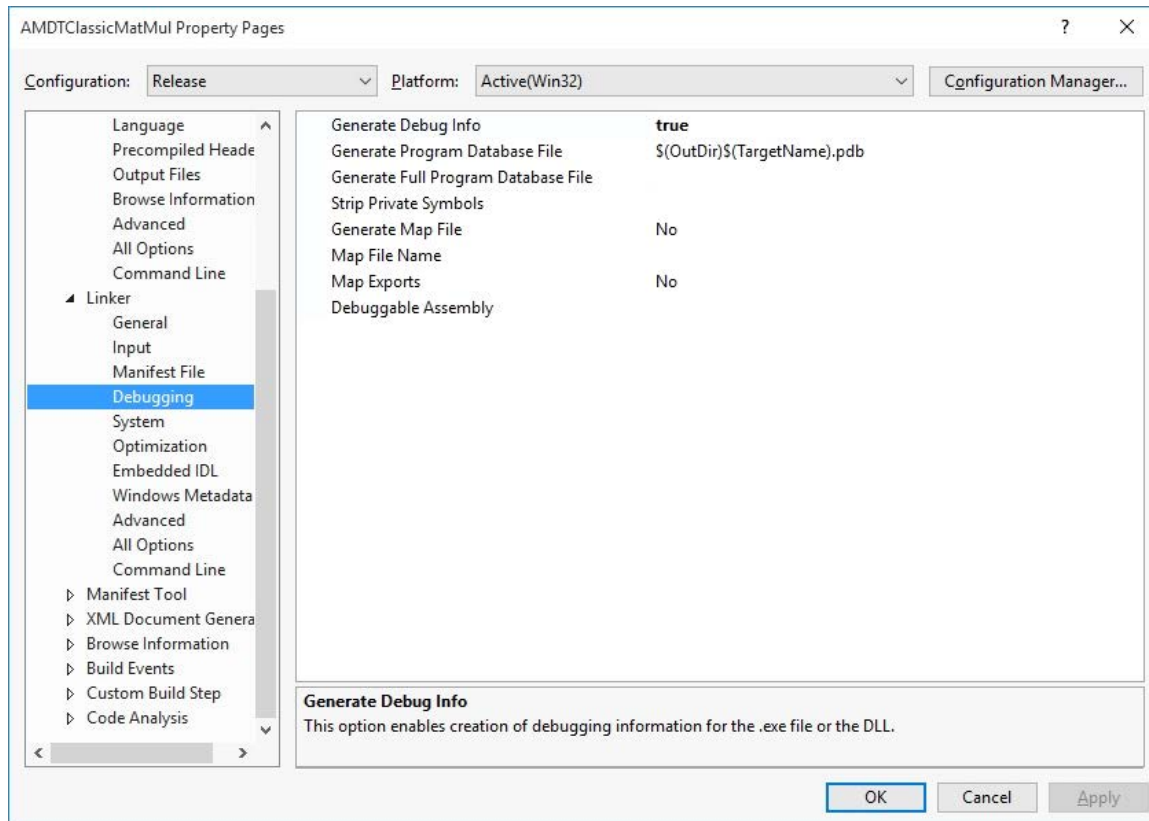


Figure 75. AMDTClassicMatMul Property Page

8. In the project pane, expand **Linker** and then select **Debugging**.
9. From the **Generate Debug Info** drop-down, select **/DEBUG**.

14.1.2 Generating 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.

14.2 CPU Profiling

The AMD uProf CPU Performance Profiling follows a sampling-based approach to gather the profile data periodically. It uses a variety of software and hardware resources available in AMD x86 based processor families. CPU Profiling uses the OS timer, HW Performance Monitor Counters (PMC), and HW IBS feature.

The following section explains the various key concepts related to CPU Profiling.

14.2.1 Hardware Sources

Performance Monitor Counters (PMC)

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

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

The number of hardware performance event counters available in each processor is implementation-dependent. For the exact number of hardware performance counters, refer the Processor Programming Reference (PPR - <https://developer.amd.com/resources/developer-guides-manuals/>) of the specific processor. . 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

The 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 the instruction fetch performance is independent of the data collected for the instruction execution performance. The support for the IBS feature is indicated by the `Core::X86::Cpuid::FeatureExtIdEc[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, the hardware events are linked with the instruction that caused them. Also, the hardware events are 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 that share L3 cache resources. All the cores in a CCX share a single L3 cache. L3PMCs are available for AMD “Zen”-based processors to monitor the performance of L3 resources. For more information, refer the respective PPR for the processor.

Data Fabric Performance Monitor Counters (DFPMC)

For AMD “Zen”-based processors, DFPMCs are available to monitor the performance of Data Fabric resources. For more information, refer the respective Processor Programming Reference (PPR) for the processor.

14.2.2 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 purpose of a 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 the data collection overhead. Since, the profile data is collected on the same system in which the workload is running, more frequent sampling increases the intrusiveness of profiling. A 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, such as Instruction Pointer, Process Id, Thread Id, and Call-stack will be collected by the interrupt handler.

Event-Counter Multiplexing

If the number of the 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 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 and thus, reduces the 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.

14.2.3 Profile Types

The profile types are classified based on the hardware or software 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 the 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. 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 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.

14.2.4 Predefined Core PMC Events

Some of the Core Performance events of AMD “Zen” processors are listed in the following table:

Table 57. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
AMD 2nd Gen EPYC™ Processors		
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, and so on.
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 mis-predicted. <i>Note: Only EX direct mis-predicts and indirect target mis-predicts are counted.</i>
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 a large increment per cycle event as it can count more than 15 events per cycle. This count both single precision and double precision FP events.

Table 57. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
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 Counts 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 Counts 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

Table 57. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
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.

Table 57. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
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. <i>Note: On AMD “Zen 3” core processors, this event counts the 64B (cache-line crossing) and 4K (page crossing) misaligned loads.</i>
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.
AMD EPYC™ 3rd Generation Processors		
0x76, 0x00	CYCLES_NOT_IN_HALT	CPU clock cycles not halted The number of CPUcycles 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, and so on.
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

Table 57. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
0xC3, 0x00	RETIRED_BR_INST_MISP	Retired Branch Instructions Mis-predicted The number of retired branch instructions, that were mis-predicted. Note that only EX direct mis-predicts and indirect target mis-predicts 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, and 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 Counts all the Instruction cache fill requests that misses in L2 cache.
0x64, 0x08	L2_CACHE_MISS.FROM_L1_DC_MIS S	L2 cache miss from L1 DC miss Counts 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.

Table 57. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
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.EXTERNAL_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.

Table 57. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
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

Table 57. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
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	The number of misaligned loads. <i>Note: On AMD “Zen 3” core processors, this event counts the 64B (cache-line crossing) and 4K (page crossing) misaligned loads.</i>
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.

The following table shows the CPU performance metrics:

Table 58. Core CPU 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	Instructions 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.

14.2.5 IBS Derived Events

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

Arithmetic calculation 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.

The following table shows the IBS fetch events:

Table 59. IBS Fetch Events

IBS Fetch Event	Description
All IBS Fetch Samples	The number of all the IBS fetch samples. This derived event counts the number of all the 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 completed IBS sampled fetches. 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. For example, 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 number of unsuccessful, speculative fetch activity. It is a lower bound as 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 (that is, address translation completed successfully) and used a 4-KByte page entry in the L1 ITLB.

Table 59. IBS Fetch Events

IBS Fetch Event	Description
IBS 2M page translation	The number of IBS attempted fetch samples where the fetch operation produced a valid physical address (that is, address translation completed successfully) and used a 2 MB 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.

The following table lists the IBS op events:

Table 60. 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 (for example, the ops that perform arithmetic and logical operations). IBS collects data for the retired ops. No data is collected for ops that are aborted due to pipeline flushes. 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 the 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.

Table 60. IBS Op Events

IBS Op Event	Description
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 (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 (that is, 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.

Table 60. IBS Op Events

IBS Op Event	Description
IBS store-to-load cancelled	The number of IBS op samples where data forwarding to a load operation from a store was canceled.
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 KB 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 MB 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 GB 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 MB 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 GB page entry for address translation.
IBS data cache miss load latency	The total DC miss load latency (in processor cycles) across all the 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	The Load Resync.

Table 60. IBS Op Events

IBS Op Event	Description
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 through 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.

14.3 Useful URLs

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

- Processor Programming Reference (PPR) for AMD 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>)