



uProf User Guide

Publication # **57368**

Revision # **4.2**

Issue Date **January 2024**

© 2024 Advanced Micro Devices Inc. All rights reserved.

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	17
About this Document	18
Intended Audience	18
Conventions	18
Abbreviations	18
Terminology	20
Part 1:	
Introduction	1
Chapter 1 Introduction	2
1.1 Overview	2
1.2 Specification	3
1.2.1 Processors	3
1.2.2 Operating Systems	3
1.2.3 Compilers and Application Environment	3
1.2.4 Virtualization Support	4
1.2.5 Container Support	4
1.3 Installing AMD uProf	5
1.3.1 Windows	5
1.3.2 Linux	5
1.3.3 FreeBSD	7
1.4 Sample Programs	7
1.5 Support	7
Part 2:	
System Analysis	8
Chapter 2 Getting started with AMDuProfPcm	9
2.1 Overview	9
2.1.1 Prerequisite(s)	9
2.2 Options	10
2.3 Commands	24

2.4	Examples	25
2.4.1	Linux and FreeBSD	25
2.4.2	Windows	26
2.5	BIOS Settings - Known Behavior	28
2.6	Monitoring without Root Privileges	28
2.7	Roofline Model	29
2.8	Pipeline Utilization	31
Chapter 3	Getting Started with AMDuProfSys	34
3.1	Overview	34
3.2	Supported Platforms	34
3.3	Supported Hardware Counters	34
3.4	Supported Operating Systems	34
3.5	Set up	35
3.5.1	Linux	35
3.5.2	Windows	35
3.6	Options	36
3.6.1	Generic	36
3.6.2	Collect Command	37
3.6.3	Report Command	38
3.7	Examples	38
3.8	Limitations	39
Part 3:	Application Analysis	41
Chapter 4	Workflow and Key Concepts	42
4.1	Workflow	42
4.1.1	Collect Phase	42
4.1.2	Translate and Report Phases	44
4.1.3	Analyze Phase	44
4.2	Predefined Sampling Configuration	44
4.3	Predefined View Configuration	46
Chapter 5	Getting Started with AMD uProf GUI	50

5.1	User Interface	50
5.2	Launching GUI	51
5.3	Configure a Profile	52
5.3.1	Select Profile Target	52
5.3.2	Select Profile Type	53
5.3.3	Advanced Options	55
5.3.4	Start Profile	57
5.4	Translation Progress	58
5.5	Analyze the Profile Data	58
5.5.1	Overview of Performance Hotspots	59
5.5.2	Thread Concurrency Graph	63
5.5.3	Function HotSpots	64
5.5.4	Process and Functions	65
5.5.5	Source and Assembly	67
5.5.6	Top-down Callstack	68
5.5.7	Flame Graph	69
5.5.8	Call Graph	70
5.5.9	IMIX View	71
5.6	Importing Profile Database	72
5.7	Analyzing Saved Profile Session	73
5.8	Using Saved Profile Configuration	74
5.9	Settings	75
5.10	Shortcut Keys	77
Chapter 6	Getting Started with AMD uProf CLI	78
6.1	Overview	78
6.2	Starting a CPU Profile	79
6.2.1	List of Predefined Sample Configurations	80
6.2.2	Profile Report	81
6.3	Starting a Power Profile	82
6.3.1	System-wide Power Profiling (Live)	82
6.4	Collect Command	83

6.4.1	Options	84
6.4.2	Windows Specific Options	87
6.4.3	Linux Specific Options	88
6.4.4	Examples	91
6.5	Report Command	94
6.5.1	Options	95
6.5.2	Windows Specific Options	97
6.5.3	Linux Specific Options	98
6.5.4	Examples	98
6.6	Translate Command	99
6.6.1	Options	100
6.6.2	Windows Specific Options	101
6.6.3	Linux Specific Options	101
6.6.4	Examples	102
6.7	Timechart Command	102
6.7.1	Options	103
6.7.2	Examples	103
6.8	Diff Command	104
6.8.1	Profile Comparison Eligibility Criteria	105
6.8.2	Options	105
6.8.3	Examples	107
6.9	Profile Command	109
6.9.1	Options	109
6.9.2	Windows Specific Options	115
6.9.3	Linux Specific Options	116
6.9.4	Examples	119
6.10	Info Command	122
6.10.1	Options	122
6.10.2	Examples	123
Chapter 7	Performance Analysis	125
7.1	CPU Profiling	125

7.2	Analysis with Time-based Profiling	127
7.2.1	Configuring and Starting Profile	127
7.2.2	Analyzing Profile Data	128
7.3	Analysis with Event-based Profiling	128
7.3.1	Configuring and Starting Profile	128
7.3.2	Analyzing Profile Data	129
7.4	Analysis with Instruction-based Sampling	130
7.4.1	Configuring and Starting Profile	130
7.4.2	Analyzing Profile Data	131
7.5	Analysis with Call Stack Samples	131
7.5.1	Flame Graph	132
7.5.2	Call Graph	133
7.6	Profiling a Java Application	134
7.6.1	Launching a Java Application	134
7.6.2	Attaching a Java Process to Profile	135
7.6.3	Java Source View	135
7.6.4	Java Call Stack and Flame Graph	136
7.7	Cache Analysis	137
7.7.1	Supported Metrics	138
7.7.2	Cache Analysis Using GUI	138
7.7.3	Cache Analysis Using CLI	139
7.8	Custom Profile	141
7.8.1	Configuring and Starting Profile	141
7.8.2	Analyzing Profile Data	144
7.9	Advisory	145
7.9.1	Confidence Threshold	145
7.9.2	Issue Threshold	145
7.10	ASCII Dump of IBS Samples	146
7.11	Branch Analysis	146
7.12	Export Session	148
7.13	Limitations	148

Chapter 8	Performance Analysis (Linux)	150
8.1	Threading Analysis	150
8.1.1	Threading Analysis Using CLI	150
8.1.2	pthread Synchronization APIs	154
8.1.3	libc System Call Wrapper APIs	154
8.1.4	Timeline Analysis GUI in Linux	156
8.2	OpenMP Analysis	160
8.2.1	Profiling OpenMP Application using GUI	161
8.2.2	Profiling OpenMP Application Using CLI	162
8.2.3	Environment Variables	164
8.2.4	Limitations	165
8.3	MPI Profiling	165
8.3.1	Collecting Data Using CLI	166
8.3.2	Analyzing the Data with CLI	167
8.3.3	Analyze the Data with GUI	168
8.3.4	Limitations	168
8.4	Profiling Support on Linux for perf_event_paranoid Values	168
8.5	Profiling Linux System Modules	169
8.6	Profiling Linux Kernel	169
8.6.1	Enabling Kernel Symbol Resolution	169
8.6.2	Downloading and Installing Kernel Debug Symbol Packages	170
8.6.3	Build Linux kernel with Debug Symbols	171
8.6.4	Analyzing Hotspots in Kernel Functions	171
8.6.5	Linux Kernel Callstack Sampling	171
8.6.6	Constraints	172
8.7	Kernel Block I/O Analysis	172
8.7.1	Kernel Block I/O Analysis Using CLI	173
8.8	GPU Offloading Analysis (GPU Tracing)	174
8.8.1	GPU Offload Analysis Using CLI	175
8.9	GPU Profiling	177
8.9.1	GPU Profiling Using CLI	178

8.10	Other OS Tracing Events	180
8.10.1	Tracing Page Faults and Memory Allocations Using CLI	180
8.10.2	Tracing Function Call Count using CLI	181
8.11	MPI Trace Analysis	182
8.11.1	MPI Light-weight Tracing Using CLI	183
8.11.2	MPI Full Tracing Using CLI	185
8.11.3	MPI FULL Tracing Using GUI	190
Chapter 9	Power Profile	195
9.1	Overview	195
9.2	Metrics	195
9.3	Using Profile through GUI	197
9.3.1	Configuring a Profile	197
9.3.2	Analyzing a Profile	198
9.4	Using CLI to Profile	199
9.4.1	Examples	200
9.5	AMDPowerProfileAPI Library	201
9.5.1	Using the APIs	201
9.6	Limitations	202
Chapter 10	Remote Profiling	203
10.1	Overview	203
10.2	Setting up Authorization	203
10.3	Launching AMDProfilerService	204
10.4	Connecting to Remote Target	205
10.5	Limitations	206
Chapter 11	AMD uProf Virtualization Support	208
11.1	OverView	208
11.2	CPU Profiling	209
11.2.1	Profiling of Guest VM from Guest VM	209
11.2.2	Profiling of Guest VM from Host System (KVM Hypervisor)	210
11.2.3	Preparing Host system to Profile Guest Kernel Modules	210
11.2.4	AMD uProf CLI with Profiling Options	210

11.2.5	Examples	211
11.3	AMDuProfPcm	212
11.4	AMDuProfSys	212
Chapter 12	Profile Control APIs	213
12.1	AMDProfileControl APIs	213
12.1.1	CPU Profile Control APIs	213
12.1.2	Using the APIs	214
12.1.3	Compiling Instrumented Target Application	215
12.1.4	Profiling Instrumented Target Application	215
12.1.5	Limitations	215
Chapter 13	Reference	216
13.1	Preparing an Application for Profiling	216
13.1.1	Generating Debug Information on Windows	216
13.1.2	Generating Debug Information on Linux	217
13.2	CPU Profiling	217
13.2.1	Hardware Sources	218
13.2.2	Profiling Concepts	219
13.2.3	Profile Types	220
13.2.4	Predefined Core PMC Events	221
13.2.5	IBS Derived Events	235
13.3	Useful URLs	248

List of Tables

Table 1.	Conventions.	18
Table 2.	Abbreviations	18
Table 3.	Terminology	20
Table 1.	User Interface	2
Table 2.	AMDuProfPcm Options	10
Table 3.	Performance Metrics for AMD EPYC™ “Zen 2”	13
Table 4.	Performance Metrics for AMD EPYC™ “Zen 3”	16
Table 5.	Performance Metrics for AMD EPYC™ “Zen 4”	20
Table 6.	AMDuProfPcm Options	24
Table 7.	Level-1 Metrics.	31
Table 8.	Level-2 Metrics.	32
Table 9.	AMDuProfSys Generic Options	36
Table 10.	AMDuProfSys Collect Command Options.	37
Table 11.	AMDuProfSys Report Command Options	38
Table 12.	Sampled Data	43
Table 13.	Predefined Sampling Configurations	44
Table 14.	Assess Performance Configurations	46
Table 15.	Threading Configuration.	46
Table 16.	Investigate Data Access Configurations	46
Table 17.	Investigate Branch Configurations	47
Table 18.	Assess Performance (Extended) Configurations.	47
Table 19.	Investigate Instruction Access Configurations	47
Table 20.	Investigate CPI Configurations.	48
Table 21.	Instruction Based Sampling Configurations	48
Table 22.	Summary Overview	60
Table 23.	Shortcut Keys	77
Table 24.	Supported Commands.	78
Table 25.	AMDuProfCLI Collect Command Options	84
Table 26.	AMDuProfCLI Collect Command – Windows Specific Options.	87

Table 27. AMDuProfCLI Collect Command – Linux Specific Options.88

Table 28. AMDuProfCLI Report Command Options.95

Table 29. AMDuProfCLI Report Command - Windows Specific Options97

Table 30. AMDuProfCLI Report Command - Linux Specific Options98

Table 31. AMDuProfCLI Translate Command Options.100

Table 32. Translate Command - Windows Specific Options101

Table 33. Translate Command - Linux Specific Options101

Table 34. AMDuProfCLI Timechart Command Options103

Table 35. AMDuProfCLI diff Command Options105

Table 36. AMDuProfCLI profile Command Options.109

Table 37. AMDuProfCLI Windows profile Command Options.115

Table 38. AMDuProfCLI Linux profile Command Options.116

Table 39. AMDuProfCLI Info Command Options.122

Table 40. AMDuProfCLI Info Command - Linux Specific Options123

Table 41. IBS OP Derived Metrics138

Table 42. Sort-by Metric.140

Table 43. Supported CPU Events158

Table 44. CPU Trace Categories158

Table 45. Support Matrix160

Table 46. MPI Profiling Support Matrix.166

Table 47. Profiling perf_event_paranoid Values on Linux.168

Table 48. I/O Operations.172

Table 49. Supported Interfaces for GPU Tracing174

Table 50. Supported Events for GPU Profiling.177

Table 51. Supported Metrics for GPU Profiling.178

Table 52. Supported Events for OS Tracing180

Table 53. Support Matrix183

Table 54. List of Supported MPI APIs for Light-weight Tracing.183

Table 55. MPI APIs.186

Table 56. Family 17h Model 00h – 0Fh (AMD Ryzen™, AMD Ryzen ThreadRipper™, and 1st Gen AMD EPYC™)195

Table 57.	Family 17h Model 10h – 1Fh (AMD Ryzen™ and AMD Ryzen™ PRO APU)	196
Table 58.	Family 17h Model 70h – 7Fh (3 rd Gen AMD Ryzen™)	196
Table 59.	Family 17h Model 30h – 3Fh (EPYC 7002)	196
Table 60.	Family 19h Model 0h – 2Fh (EPYC 7003 and EPYC 9000)	197
Table 61.	AMDProfilerService Options	204
Table 62.	AMD uProf Virtualization Support.	208
Table 63.	AMD uProf CLI Collect Command Options	210
Table 64.	Predefined Core PMC Events	221
Table 65.	Core CPU Metrics	233
Table 66.	IBS Fetch Events.	236
Table 67.	IBS Fetch Metrics	238
Table 68.	IBS Op Events.	239
Table 69.	IBS Op Metrics for AMD “Zen4” and AMD “Zen3” Server Platforms.	245

List of Figures

Figure 1.	Sample Roofline Chart	31
Figure 2.	Sample Report.	32
Figure 3.	AMD uProf GUI.	50
Figure 4.	AMD uProf Welcome Screen.	51
Figure 5.	Start Profiling - Select Profile Target	53
Figure 6.	Start Profiling - Select Profile Configuration	54
Figure 7.	Start Profiling - Advanced Options 1	55
Figure 8.	Start Profiling - Advanced Options 2	56
Figure 9.	Profile Data Collection	57
Figure 10.	Translation Progress	58
Figure 11.	Summary - Hot Spots Screen	59
Figure 12.	OS Trace	61
Figure 13.	GPU Trace	61
Figure 14.	Summary - Thread Concurrency Graph	63
Figure 15.	ANALYZE - Function Hotspots.	64
Figure 16.	Analyze - Metrics	65
Figure 17.	SOURCES - Source and Assembly	67
Figure 18.	Top-down Callstack	68
Figure 19.	ANALYZE - Flame Graph	69
Figure 20.	ANALYZE - Call Graph.	70
Figure 21.	IMIX View	71
Figure 22.	Import Session – Importing Profile Database.	72
Figure 23.	PROFILE - Recent Session(s)	73
Figure 24.	PROFILE - Saved Configurations	74
Figure 25.	SETTINGS - Preferences	75
Figure 26.	SETTINGS - Symbols	75
Figure 27.	SETTINGS - Source Data.	76
Figure 29.	Collect and Report Commands.	79
Figure 30.	Supported Predefined Configurations on Linux	80

Figure 31.	Supported Predefined Configurations on Windows	81
Figure 32.	Output of timechart --list Command.	83
Figure 33.	Execution of timechart	83
Figure 34.	Time-based Profile – Configure	127
Figure 35.	Event-based Profile – Configure.	129
Figure 36.	IBS Configuration.	130
Figure 37.	Start Profiling - Advanced Options	132
Figure 38.	ANALYZE - Flame Graph	133
Figure 39.	ANALYZE - Call Graph.	134
Figure 40.	Java Method - Source View	136
Figure 41.	Java Application - Flame Graph	137
Figure 42.	Cache Analysis	139
Figure 43.	Cache Analysis - Summary Sections	140
Figure 44.	Cache Analysis - Detailed Report.	140
Figure 45.	CPU Trace.	142
Figure 46.	GPU Trace	143
Figure 47.	Custom Config - Added Categories	144
Figure 48.	CPI Metric - Threshold-based Performance	145
Figure 49.	Branch Analysis Summary	147
Figure 50.	Trace Report	152
Figure 51.	Timeline Analysis GUI in Linux	157
Figure 52.	Enable OpenMP Tracing	161
Figure 53.	HPC - Overview	161
Figure 54.	HPC - Parallel Regions.	162
Figure 55.	An OpenMP Report	163
Figure 56.	Disk I/O Summary Tables	173
Figure 57.	ANALYZE - Block I/O Stats	174
Figure 58.	GPU Tracing Report.	176
Figure 59.	GPU Profile Report.	179
Figure 60.	Pagefault and Memory Allocation Summary	181
Figure 61.	Function Count Summary.	181

Figure 62. LWT Report185

Figure 63. MPI Communicator Summary Table188

Figure 64. MPI Rank Summary Table188

Figure 65. MPI API Summary Table189

Figure 66. MPI Communication Matrix.....189

Figure 67. MPI Collective API Summary Table189

Figure 68. Import Profile Session190

Figure 69. MPI Communication Matrix.....190

Figure 70. MPI Rank Timeline191

Figure 71. MPI P2P API Summary192

Figure 72. MPI Collective API Summary192

Figure 73. Live System-wide Power Profile198

Figure 74. Timechart Page199

Figure 75. --list Command Output.....200

Figure 76. Timechart Run200

Figure 77. Client ID203

Figure 78. Remote Profiling Connection Establishment204

Figure 79. Selecting IP.....205

Figure 80. Connect to Remote Machine.....205

Figure 81. Remote Target Data206

Figure 82. Disconnect Button.....206

Figure 83. AMDTClassicMatMul Property Page217

Revision History

Date	Revision	Description
January 2024	4.2	Made some minor edits and updates
August 2023	4.1	Included AMD uProf 4.1 features
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://www.amd.com/en/developer/uprof.html>).

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
APERF	Actual Performance Frequency Clock Counter
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

Table 2. Abbreviations

Abbreviation	Description
CLI	Command Line Interface
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
MPERF	Maximum Performance Frequency Clock Counter
MSR	Model Specific Register
NB	Northbridge
OS	Operating System
P0Freq	P0 State Frequency
PMC	Performance Monitoring Counter
PTI	Per Thousand Instructions
RAPL	Running Average Power Limit
SMU	System Management Unit
TBP	TimeBased Profiling
TSC	Time Stamp Counter
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.
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.

Part 1:

Introduction

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.

AMD uProf has the following user interfaces:

Table 1. 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 to the AMD uProf Release Notes available at:

<https://www.amd.com/en/developer/uprof.html>

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 (Time 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™ 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 C/C++ and Fortran 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 208. The following virtualized environments are supported:

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

1.2.5 Container Support

AMD uProf CPUProfiler can be used for analysis of applications running inside the Docker container environments. This is supported only on Linux platforms. Choose one of the following approaches for application analysis:

- Run AMD uProf inside the Docker container to analyze the application. `CAP_SYS_ADMIN` permission (`docker run --cap-add=CAP_SYS_ADMIN`) is required to enable profiling. Both CLI and GUI based profiling and analysis supported in this mode.

- Run AMD uProf CLI outside the Docker container to profile and analyze the target application running in the container:
 - Attach uProf CLI to the containerized process using the `--pid` option during collection. Alternatively, collect the system-wide data and filter by PID during report generation.
 - During report generation, provide the path to the binary and source code (`--bin-path` and `--src-path`) of the profiled application running in the container. AMD uProf GUI doesn't support profiling and analysis in this mode.

1.3 Installing AMD uProf

Download the latest version of the AMD uProf installer package for the supported operating systems from the AMD portal (<https://www.amd.com/en/developer/uprof.html>). 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

1.3.2.1 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.*

1.3.2.2 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/*.

1.3.2.3 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/*.

1.3.2.4 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
```

1.3.2.5 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.

1.3.2.6 Installing ROCm

Complete the steps in the ROCm installation guide (https://docs.amd.com/bundle/ROCm-Installation-Guide-v5.5/page/Introduction_to_ROCm_Installation_Guide_for_Linux.html) to install AMD ROCm™ v5.5 on the host system.

After ROCm 5.5 installation, make sure symbolic link of `/opt/rocm/` points to `/opt/rocm-5.5.0/`.

```
$ ln -s /opt/rocm-5.5.0/ /opt/rocm/
```

AMD ROCm v5.5 installation is required for GPU tracing and profiling.

1.3.2.7 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 the AMD portal (<https://www.amd.com/en/developer/uprof.html>).

Part 2:

System Analysis

Chapter 2 Getting started with AMDuProfPcm

2.1 Overview

The System Analysis utility AMDuProfPcm helps to monitor basic performance monitoring metrics for AMD EPYC™ 7001, AMD EPYC™ 7002, AMD EPYC™ 7003, and AMD EPYC™ 9000 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.

2.1.1 Prerequisite(s)

2.1.1.1 Linux

- AMDuProfPcm requires the MSR driver and either root privileges or read write permissions for *dev/cpu/*/msr* devices only when it is used with `--msr` for data collection.
- 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
```
- Roofline plotting script (*AMDuProfModelling.py*) requires python 3.x and python module 'matplotlib'

2.1.1.2 FreeBSD

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

Synopsis:

```
AMDuProfPcm [<COMMANDS>] [<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/pmcddata.txt
# AMDuProfPcm -m memory -a -d 10 -o /tmp/memdata.txt -- /tmp/myapp.exe
```

2.2 Options

The following table lists all the options:

Table 2. 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” and AMD “Zen4” 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) • swpfdc – software prefetch data cache from various nodes and CCX (supported only on AMD “Zen3” and AMD “Zen4” processors) • hwpfdc – hardware prefetch data cache from various nodes and CCX (supported only on AMD “Zen3” and AMD “Zen4” processors) • pipeline_util – top-down metrics to visualize the bottlenecks in the CPU pipeline (supported only on AMD “Zen4” processors)

Table 2. AMDuProfPcm Options

Option	Description
-c <core ccx l3 ccd package>=<n>	<p>Collect from the specified core ccx ccd package. The default is 'core=0'.</p> <p>If 'ccx' or 'l3' 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 2. 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. Notes: <ol style="list-style-type: none"> Options -i and -m cannot be used together. If option -i is used, all the events mentioned in the user defined config file will be collected.
-d <seconds>	Profile duration to run.
-t < multiplex interval in ms>	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. Note: This is supported only on Linux.
-w <dir>	Specifies the working directory. The default will be the path of the launched application.
-v	Print version.
-X	Collect data using perf subsystem without root privileges. Note: This is only supported on Linux.
-P <process ID>	Specify the target process ID to monitor. Note: This is only supported with the option -X on Linux.
-f <util:<n>>	Filter the roofline data based on the utilization. For example, -f util:90 will filter all data points with less than 90% utilization. Note: This is a applicable only with the roofline command.

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

Table 3. 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 $CEF = (APERF / TSC) * P0Freq$. APERF is incremented in proportion to the actual number of core cycles while the core is in C6 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 3. 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 3. 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 4. 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 $CEF = (APERF / TSC) * P0Freq$. APERF is incremented in proportion to the actual number of core cycles while the core is in C6 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 4. 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 4. 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.

Table 4. Performance Metrics for AMD EPYC™ “Zen 3”

Metric Group	Metric	Description
swpfdc	SwPf DC Fills from DRAM or IO connected in remote node (pti) SwPf DC Fills from CCX Cache in remote node (pti) SwPf DC Fills from DRAM or IO connected in local node (pti) SwPf DC Fills from Cache of another CCX in local node (pti) SwPf DC Fills from L3 or different L2 in same CCX (pti) SwPf DC Fills from L2 (pti)	Software prefetch data cache from various nodes and CCX.
hwpfdc	HwPf DC Fills from DRAM or IO connected in remote node (pti) HwPf DC Fills from CCX Cache in remote node (pti) HwPf DC Fills from DRAM or IO connected in local node (pti) HwPf DC Fills from Cache of another CCX in local node (pti) HwPf DC Fills from L3 or different L2 in same CCX (pti) HwPf DC Fills From L2 (pti)	Hardware prefetch data cache from various nodes and CCX.

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

Table 5. 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 $CEF = (APERF / TSC) * P0Freq$. APERF is incremented in proportion to the actual number of core cycles while the core is in C6 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 5. 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 5. 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) Local DRAM Write Data Bytes (GB/s)	DRAM read and write data bytes for a local processor.
	Remote DRAM Read Data Bytes (GB/s) Remote DRAM Write Data Bytes (GB/s)	DRAM read and write data bytes for a remote processor.
	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 (not available in AMD “Zen1”, AMD “Zen2”, and AMD “Zen3” processors)	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 5. Performance Metrics for AMD EPYC™ “Zen 4”

Metric Group	Metric	Description
pcie	PCIe0 (GB/s) PCIe1 (GB/s) PCIe2 (GB/s) PCIe3 (GB/s)	Approximate PCIe bandwidth in GB/s.
swpfdc	SwPf DC Fills from DRAM or IO connected in remote node (pti) SwPf DC Fills from CCX Cache in remote node (pti) SwPf DC Fills from DRAM or IO connected in local node (pti) SwPf DC Fills from Cache of another CCX in local node (pti) SwPf DC Fills from L3 or different L2 in same CCX (pti) SwPf DC Fills from L2 (pti)	Software prefetch data cache from various nodes and CCX.
hwpfdc	HwPf DC Fills from DRAM or IO connected in remote node (pti) HwPf DC Fills from CCX Cache in remote node (pti) HwPf DC Fills from DRAM or IO connected in local node (pti) HwPf DC Fills from Cache of another CCX in local node (pti) HwPf DC Fills from L3 or different L2 in same CCX (pti) HwPf DC Fills From L2 (pti)	Hardware prefetch data cache from various nodes and CCX.

Table 5. Performance Metrics for AMD EPYC™ “Zen 4”

Metric Group	Metric	Description
pipeline_util	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.	

2.3 Commands

The following table lists all the commands:

Table 6. AMDuProfPcm Options

Command	Description
roofline	Collects data required for generating roofline model.

2.4 Examples

2.4.1 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 data from cores 0-7 and run the application on cores 0-3:

```
# ./AMDuProfPcm -m ipc -c core=0-7 -o /tmp/pcmdata.csv -- /usr/bin/taskset -c 0-3 <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
```
- Collect roofline data in root mode:

```
sudo ./AMDuProfPcm roofline -o /tmp/roofline.csv <application>
```
- Collect roofline data in non-root mode:

```
./AMDuProfPcm roofline -X -o /tmp/roofline.csv <application>
```
- Plot roofline data and generate a PDF in the output directory */tmp*:

```
AMDuProfModelling.py -i /tmp/roofline.csv -o /tmp/
```

2.4.2 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
```

2.5 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

2.6 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 pipeline_util -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 pipeline_util -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
```

2.7 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:

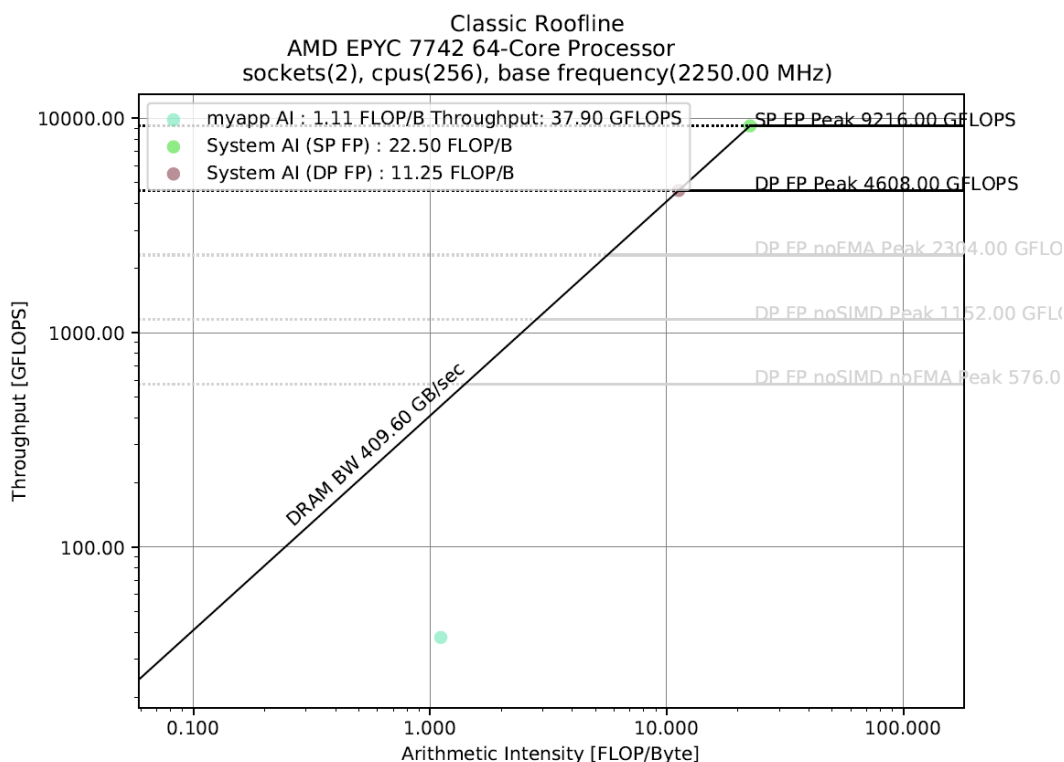


Figure 1. Sample Roofline Chart

2.8 Pipeline Utilization

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

The level-1 metrics are as follows:

Table 7. 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 8. 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 pipeline_util -c core=0 -A system -o /tmp/myapp-td.csv -- /usr/bin/taskset -c 0 myapp.exe
```

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

```
$ AMDuProfPcm -X -m pipeline_util -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_Spec	Backend_Bound.Memory	Backend_Bound.CPU
31528583352	0	0.18	0	33.19	66.39	3.85	0.18	0	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 pipeline_util -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 pipeline_util -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 pipeline_util -a -A system -C -o /tmp/td.csv -- /tmp/myapp.exe
```

Chapter 3 Getting Started with AMDuProfSys

3.1 Overview

AMDuProfSys is a python-based system analysis tool for AMD processors. It can be used to collect the hardware events and evaluate the simple counter values or complex recipes using collected raw events. The performance metrics are based on the profile data collected using Core, L3, DF, and UMC PMCs. This tool can be used to get the overall performance details of the hardware blocks used in the system.

3.2 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
- Family 19, model 0xA0 - 0xAF

3.3 Supported Hardware Counters

- CORE PMC
- DF PMC
- L3 PMC
- UMC PMC

3.4 Supported Operating Systems

- Linux
- Windows

3.5 Set up

Follow the installation steps in the section *"Installing AMD uProf" on page 5*.

3.5.1 Linux

If tar ball is used, uProf driver must be used manually. If you are not using uProf driver, optionally, you can use Linux perf. However, you must ensure that Linux user space tool is installed and Perf tools support the required PMC event monitoring. If uProf driver is not used, command line must include the option `--use-linux-perf`.

To install user space perf tool:

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

NMI watchdog must be disabled, this requires root privileges:

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

Perf parameter should be set to -1 if system-wide profile data or DF and L3 metrics must be collected:

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

3.5.2 Windows

Setup file will install all the required components to run AMDuProfSys.

After installation, AMDuProfSys is available in the following directory:

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

Python Packages

AMDuProfSys requires Python to be installed on the target platform. Supported minimum Python version is 3.6. When the tool is executed for the first time, it will prompt to install the following Python modules:

- `tqdm` — use `pip3 install tqdm` to install
- `xlsxwriter` — use `pip3 install XlsxWriter` to install
- `yaml` — use `apt-get install python-yaml` OR `pip3 install pyyaml` to install
- `yamlordereddictloader` — use `pip3 install yamlordereddictloader` to install
- `rich` — use `pip3 install rich` 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
```
- Default metrics (core, L3 and DF) collection:

```
AMDuProfSys.py -o default -a -d 100
```
- Collect and report all counters together:

```
AMDuProfSys.py --config core,l3,df,umc -o all -a -d 100
```
- Collect any user defined custom metrics from command line:

```
./AMDuProfSys.py --metrics core/BrMisPredExTime="(0x4300C3)/(0x4300C2)",core/ratio="((BrMisPredExTime * 0x430076)/0x4300C0)" -d 20
```
- 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
```
- Time series profile data for core metrics (core 0-5) with an interval of 1000 ms and set affinity of running application to core 0:

```
AMDuProfSys.py --config core -C 0-5 -I 1000 --use-linux-perf -T -o output --affinity 0 <application>
```

Note: Time series profile data collection is available only with the option -use-linux-perf.
- Collect metrics for CORE, L3, DF and UMC metrics together:

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

3.6 Options

3.6.1 Generic

The following table lists the generic options:

Table 9. 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)

Table 9. AMDuProfSys Generic Options

Option	Description
--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

3.6.2 Collect Command

The following table lists the collect command options:

Table 10. 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>
-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	Collect events using raw events file. The report can be generated only at AMD. This option helps collect bigger set of metrics.
--use-linux-perf	This option can be used in Linux to collect the profile data using Linux Perf instead of AMDuProf driver.
-m --metrics	Collect user defined custom metrics through command line.

Table 10. AMDuProfSys Collect Command Options

Option	Description
--affinity	Comma separated list of CPUs. Workload is run on the configured CPUs.

3.6.3 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 11. AMDuProfSys Report Command Options

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

3.7 Examples

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

```
AMDuProfSys.py --config core -a sleep 50
```

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

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

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

```
AMDuProfSys.py --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.

3.8 Limitations

- UMC profiling is not available in Linux for the following platforms:
 - Family 17, model 0x30 - 0x3F
 - Family 19, model 0x0 - 0xF
- Time series profile data collection is available only in Linux using the option `--use-linux-perf`.

Part 3:

Application Analysis

Chapter 4 Workflow and Key Concepts

4.1 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.

4.1.1 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
- CPU Trace
- GPU Profile
- GPU Trace
- System-wide Power Profile

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 12. 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
CPU Tracing	<ul style="list-style-type: none"> • User mode trace — Collects syscall and pthread data • OS trace — Collects schedule, diskio, syscall, pthread, and funccount data 	Not applicable
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 46.

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, uProf 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 when CLI generates the profile report or GUI generates the visualization.

4.1.2 Translate and Report Phases

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.

4.1.3 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 44.

4.2 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 13. 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 13. 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.
	Threading Analysis	threading	To get an overall threading analysis and find potential issues for further investigation. <i>Note: This configuration is available only on Linux. It is supported only on AMD "Zen3" and AMD "Zen4" processors.</i>
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.

Notes:

1. The AMDuProf GUI uses the **name** of the predefined configuration in the above table.
2. The abbreviation (in Table 13 on page 44) 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.

4.3 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 14. Assess Performance Configurations

View Configuration	Abbreviation	Description
Assess Performance	trriage_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.

Following table lists the threading configuration:

Table 15. Threading Configuration

View Configuration	Abbreviation	Description
IPC assessment	ipc_assess	Find hot spots with low instruction level parallelism, it provides performance indicators – IPC and CPI. <i>Note: This configuration is available only on Linux. It is supported only on AMD “Zen3” and AMD “Zen4” processors.</i>

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

Table 16. 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.

Table 16. Investigate Data Access Configurations

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 17. 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.
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 18. 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 19. 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.

Table 19. Investigate Instruction Access Configurations

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 20. 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 21. 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.

Table 21. Instruction Based Sampling Configurations

View configuration	Abbreviation	Description
IBS BR branch	ibs_op_branch	You can use this view to display the IBS retired branch op measurements including mis-predicted and taken branches.
IBS BR return	ibs_op_return	You can use this view to display the IBS return op measurements including the return mis-prediction 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.
New Views in AMD “Zen3” and AMD “Zen4” Processors		
IBS Load Op Analysis	ibs_op_ld	You can use this view to analyze the memory load performance issues of an application.
IBS Load Op Analysis (ext)	ibs_op_ld_ext	You can use this view to analyze the memory load performance issues of an application.
IBS Branch Overview	mibs_op_br_overview	You can use this view to analyze the branch metrics.
IBS Load Latency Analysis	ibs_op_ld_lat	You can use this view to analyze the memory load latency performance issues of an application.
IBS Memory Overview	ibs_op_ls_overview	You can use this view to understand the memory access pattern of an application.
IBS Perf Overview	ibs_op_overview	You can use this view to understand the performance characteristics of an application.

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 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:

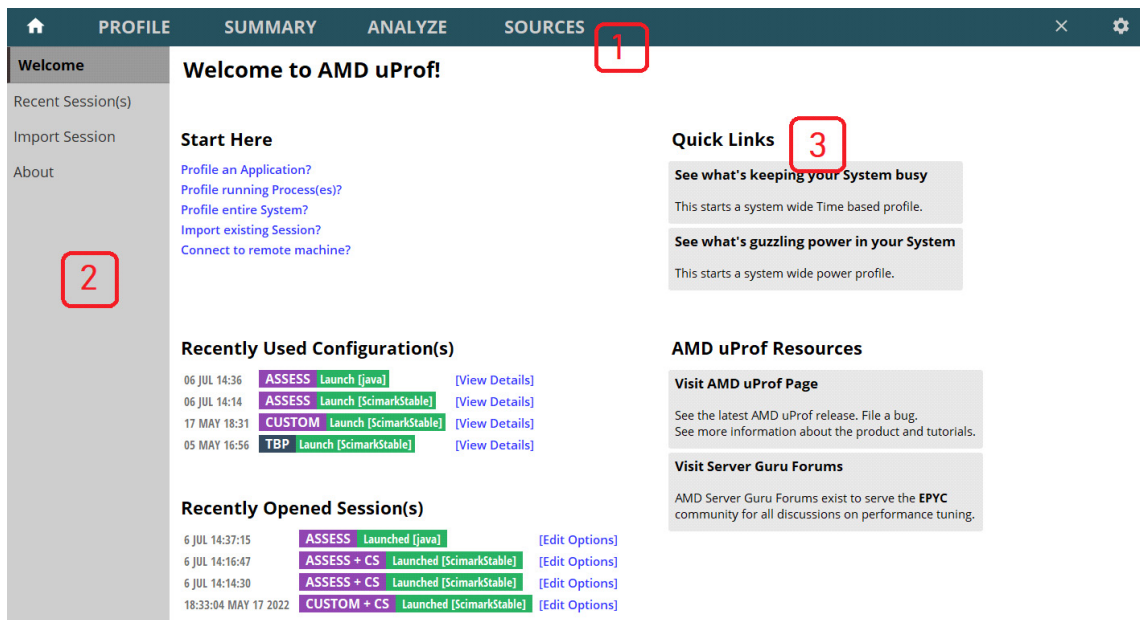


Figure 3. AMD uProf GUI

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:

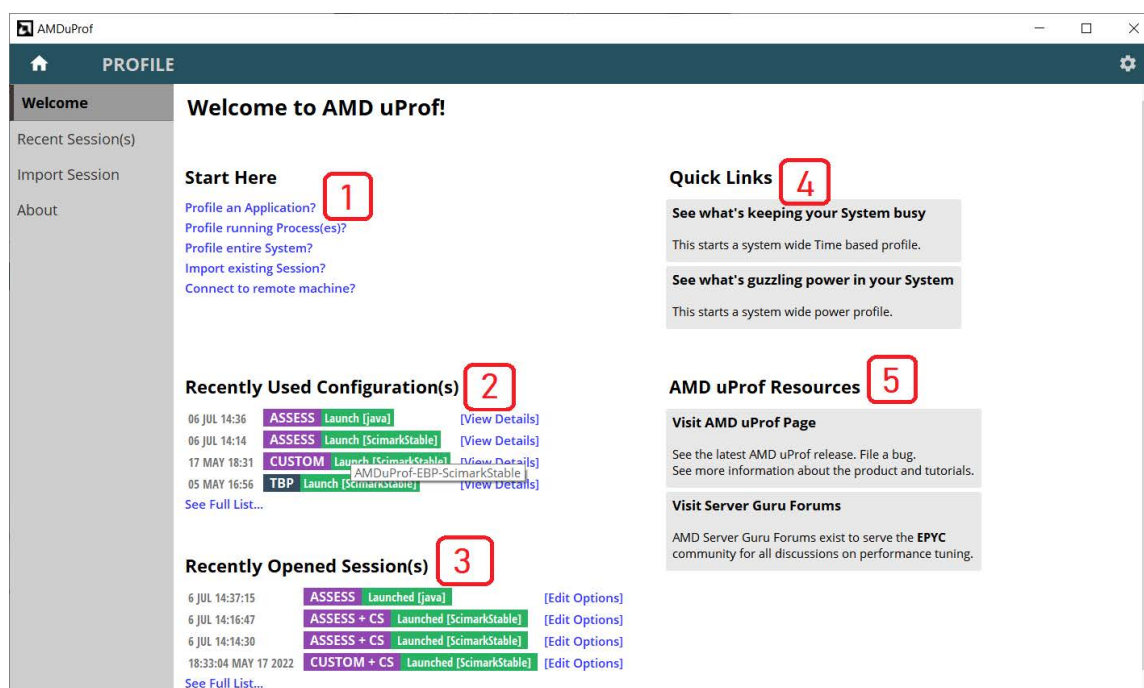


Figure 4. AMD uProf Welcome Screen

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, CPU Trace, GPU Trace, or 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 43 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:

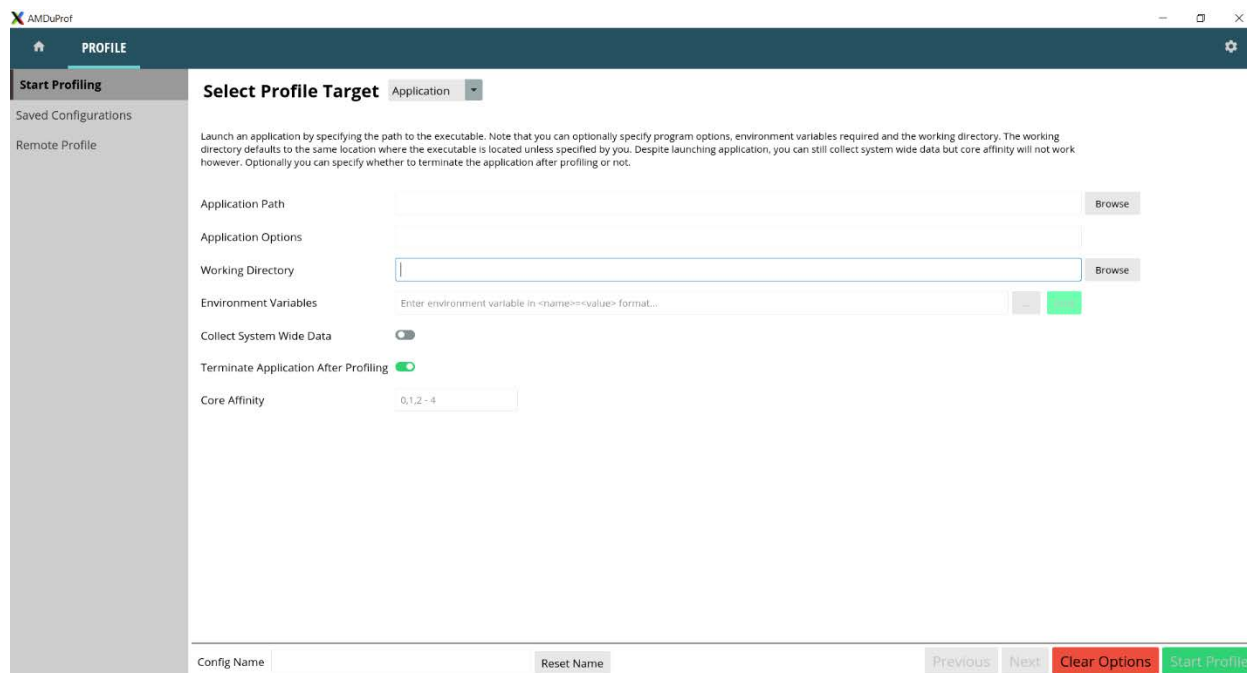


Figure 5. Start Profiling - Select Profile Target

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 Configuration** screen is displayed as follows:

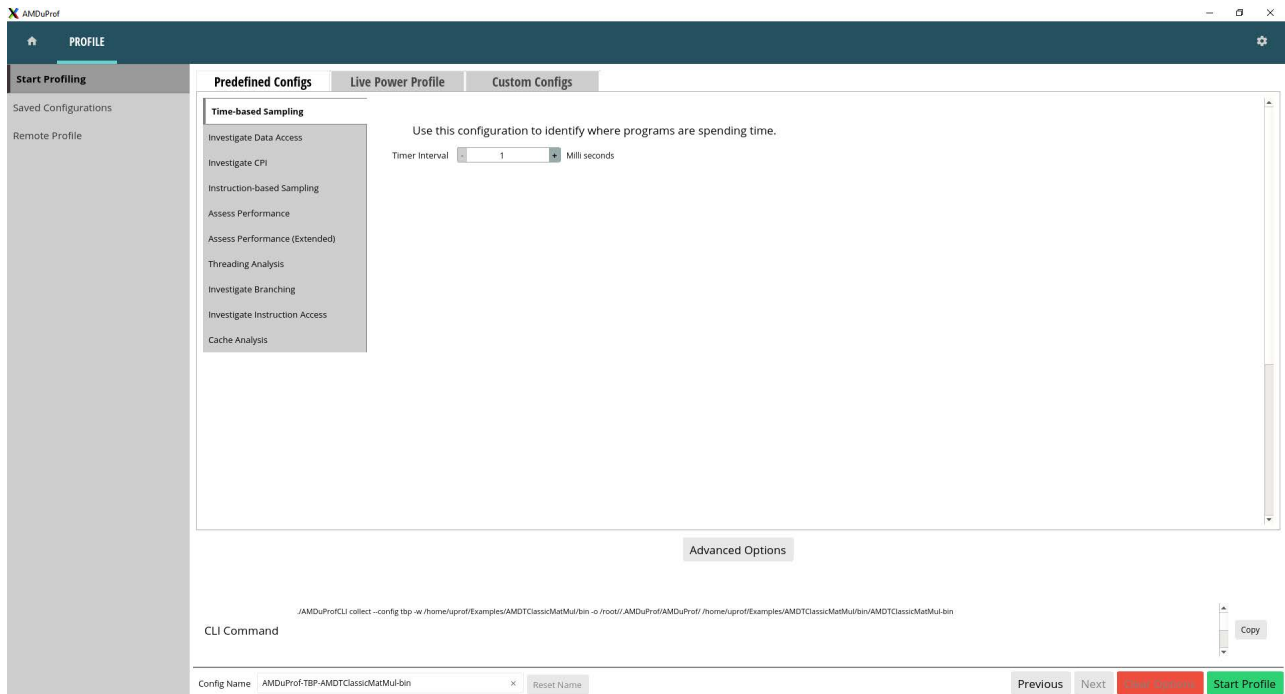


Figure 6. Start Profiling - Select Profile Configuration

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 one of the following tabs:
 - **Predefined Configs** consists of all the predefined configurations, such as **Time-based Profiling**, **Cache Analysis**, and **Assess Performance**.
 - Live Power Profiling consists of options to perform real-time power profiling.
 - **Custom Configs** has options to perform **Custom CPU Profile**, **CPU Tracing**, and **GPU Tracing**.
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. Modify event options are available only for the predefined configurations.
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 43 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.

The CLI command is available at the bottom of this page, which displays the CLI version of the GUI option selected on the **Select Profile Configuration** page.

5.3.3 Advanced Options

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

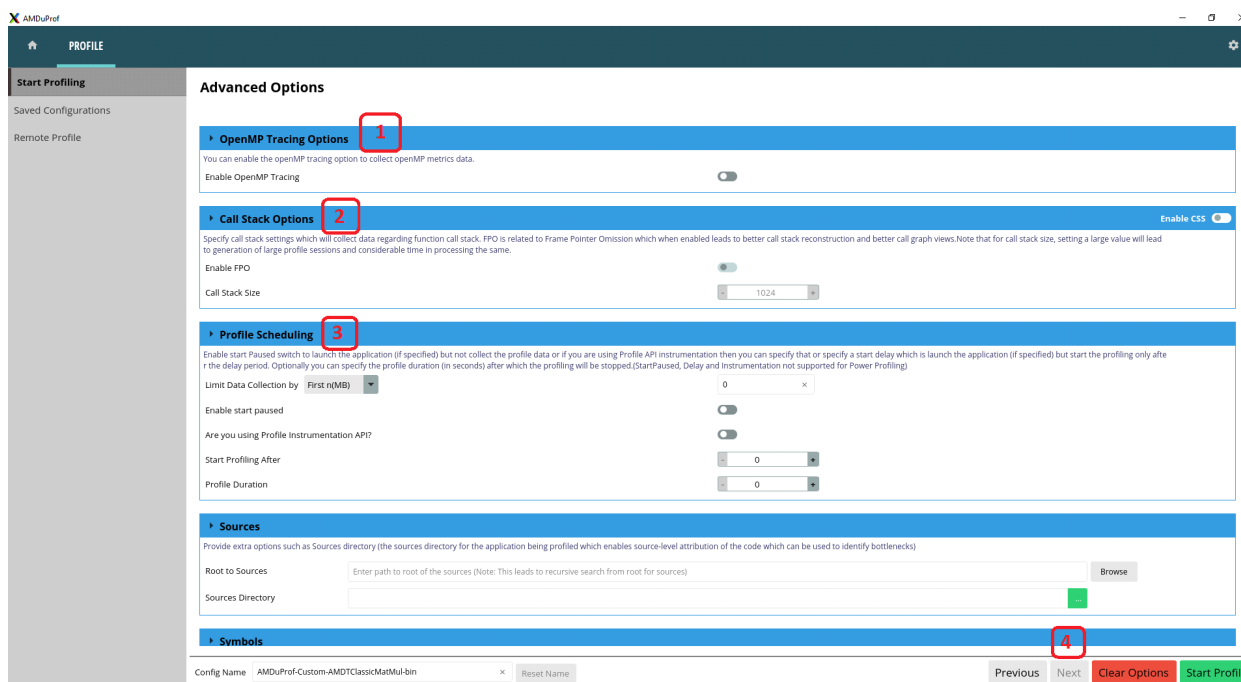


Figure 7. Start Profiling - Advanced Options 1

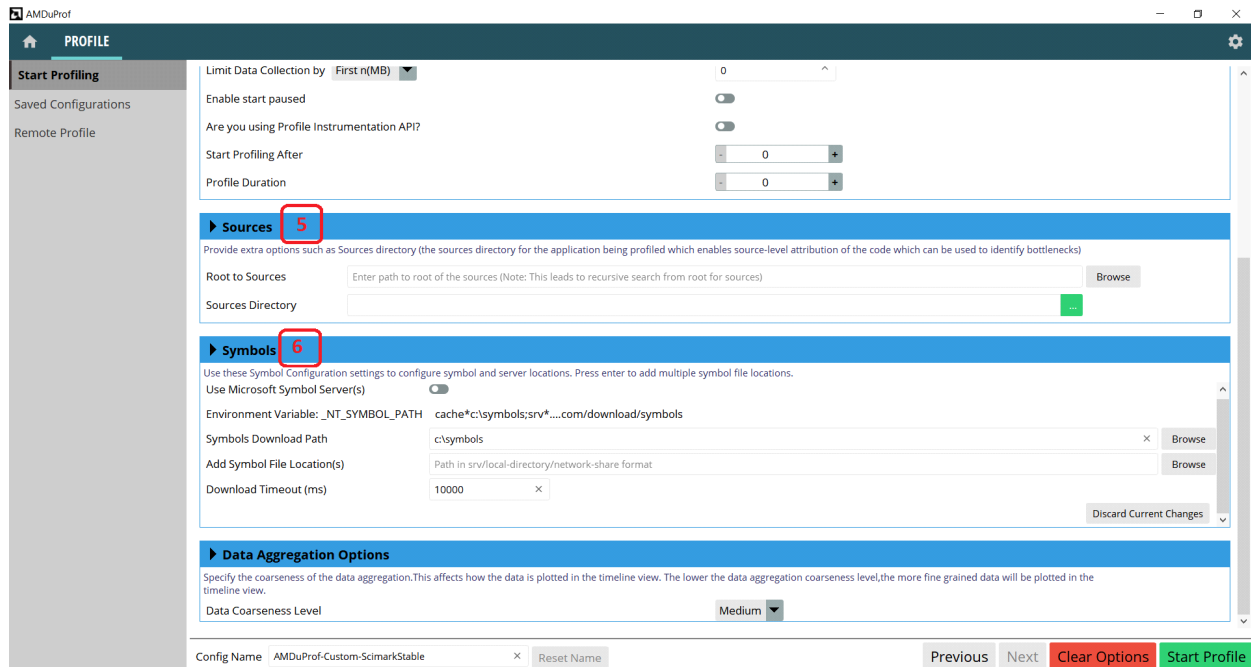


Figure 8. Start Profiling - Advanced Options 2

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 **Top-Down Callstack**, **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.

You can also provide **Download timeout** for symbol file download from the server.

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:

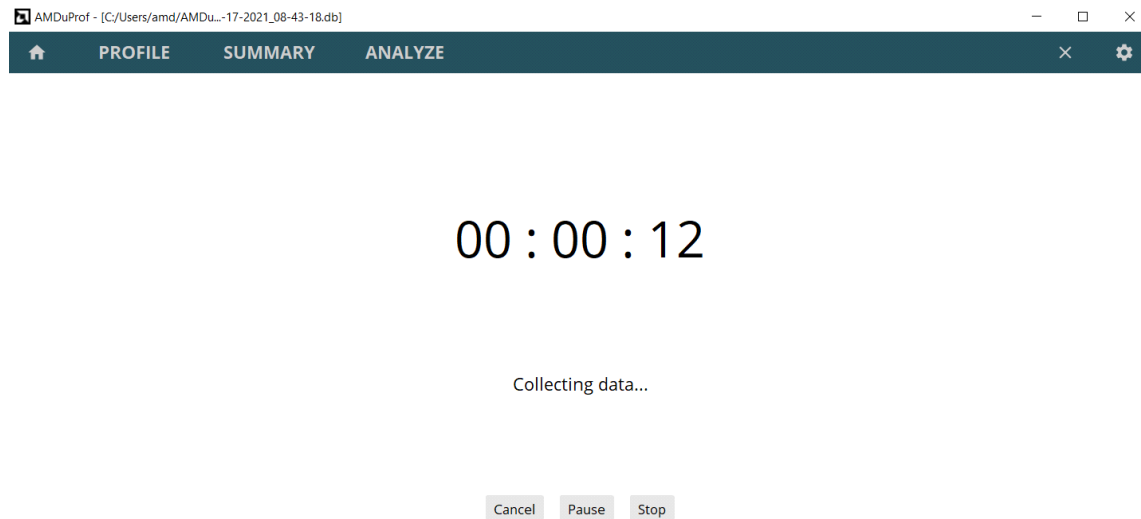


Figure 9. Profile Data Collection

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 Translation Progress

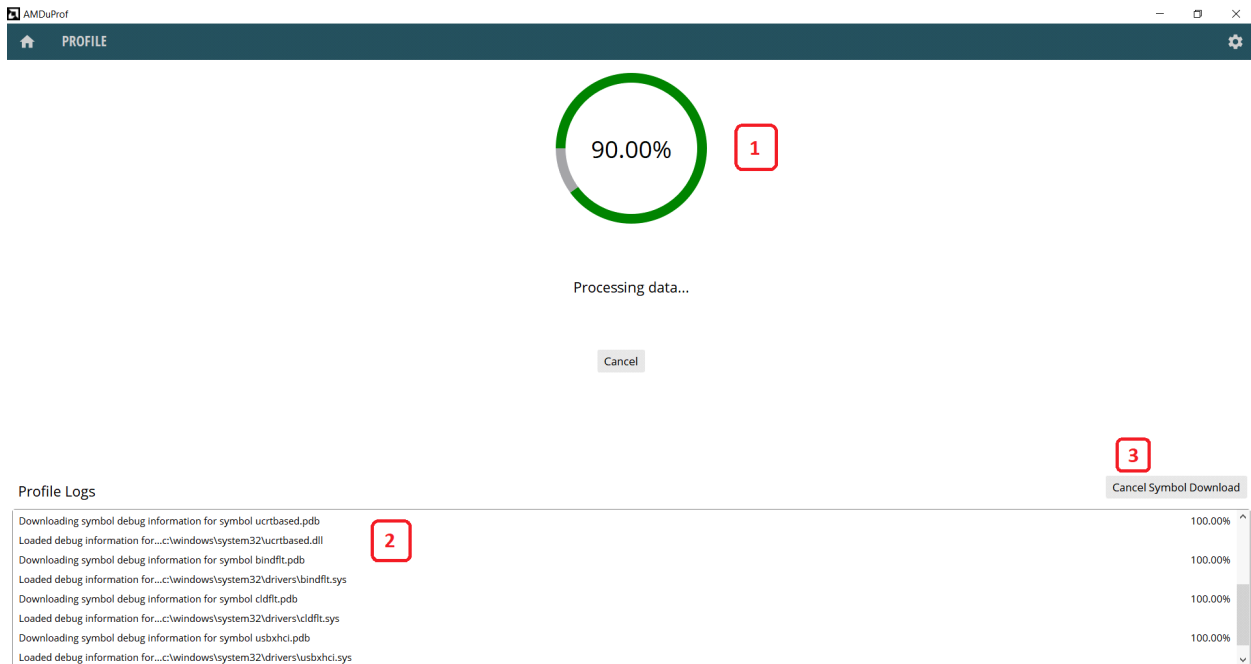


Figure 10. Translation Progress

This screen displays:

1. The percentage of translation completed.
2. Profile Logs display the currently loaded symbol and the corresponding download percentage it is being downloaded.
3. **Cancel Symbol Download** button to stop the symbol download from the server provided in the symbol settings page.

Note: This option is available only on Windows.

5.5 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 information as a timeline chart.

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

5.5.1 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:

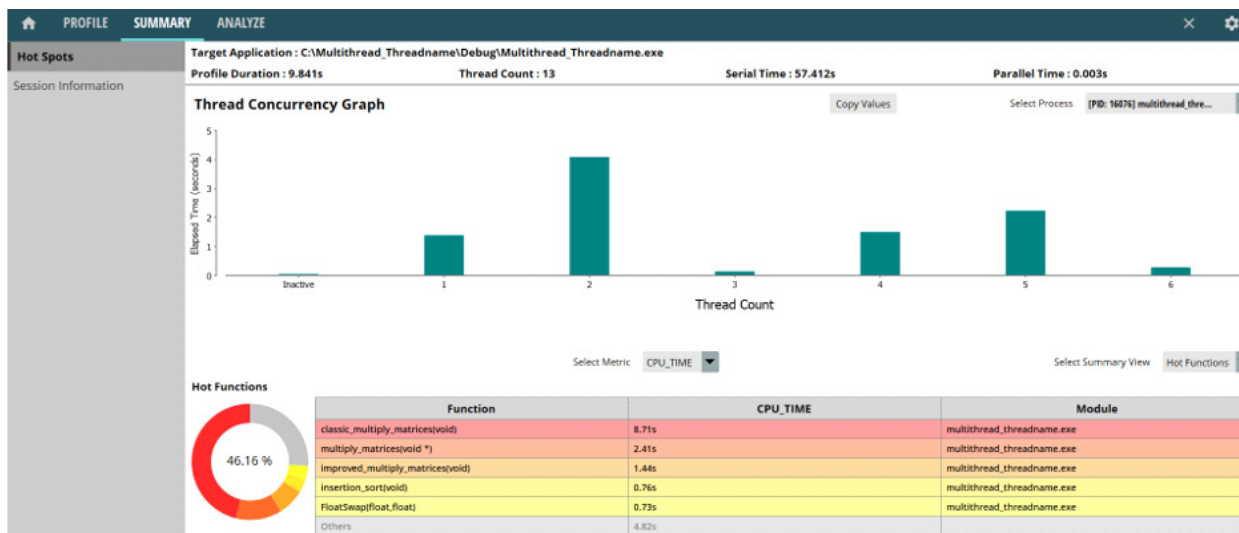


Figure 11. Summary - Hot Spots Screen

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.

4. From the **Select Summary View** drop-down, select one of the following:

- **Hot Threads**
- **Hot Processes**
- **Hot Functions**
- **Hot Modules**

Based on the selection, one donut will be displayed at a time.

Summary Overview

Based on the selection, the **Summary Overview** screen will look similar to the following:

Table 22. Summary Overview

Data Collected	Table Present	Description	Timing Details
OS Trace	Schedule Summary	Summary of per thread running/wait time (percentages).	<ul style="list-style-type: none"> • Profile Duration • Parallel Time • Serial Time • Wait Time • Sleep Time
	Wait Object Summary	Time spent in operations related to several types of synchronization objects, that is, locks, mutexes, condition variables, and so on.	
	Wait Function Summary	Time spent in several types of pthread blocking functions, that is, pthread_join, and so on.	
	Syscall Summary	Time spent in syscall(s)	
GPU Trace	GPU Kernel Summary	Time spent per GPU kernel in execution in the enqueued device.	<ul style="list-style-type: none"> • Profile Duration
	Data Transfer Summary	Time spent in GPU data copy operations.	
MPI Trace	MPI P2P API Summary	Time spent in various MPI P2P API across all ranks of the profile.	<ul style="list-style-type: none"> • Profile Duration • Parallel Time • Serial Time • MPI Time
	MPI Collective API Summary	Time spent in various MPI collective communication API across all ranks of the profile.	
CPU Profile	Hot Functions	Hottest functions based on CPU profile.	<ul style="list-style-type: none"> • Profile Duration • Parallel Time • Serial Time
	Hot Modules	Hottest modules based on CPU profile.	
	Hot Threads	Hottest threads based on CPU profile.	
	Hot Processes	Hottest processes based on CPU profile.	

OS Trace

The OS Trace screen will look as follows:

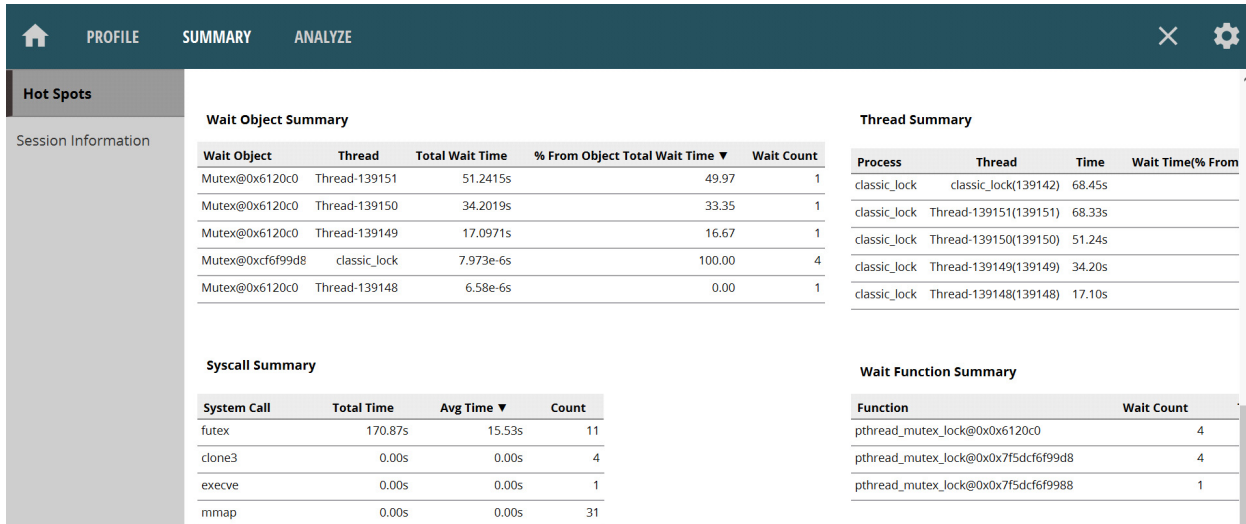


Figure 12. OS Trace

GPU Trace

The GPU Trace screen will look as follows:

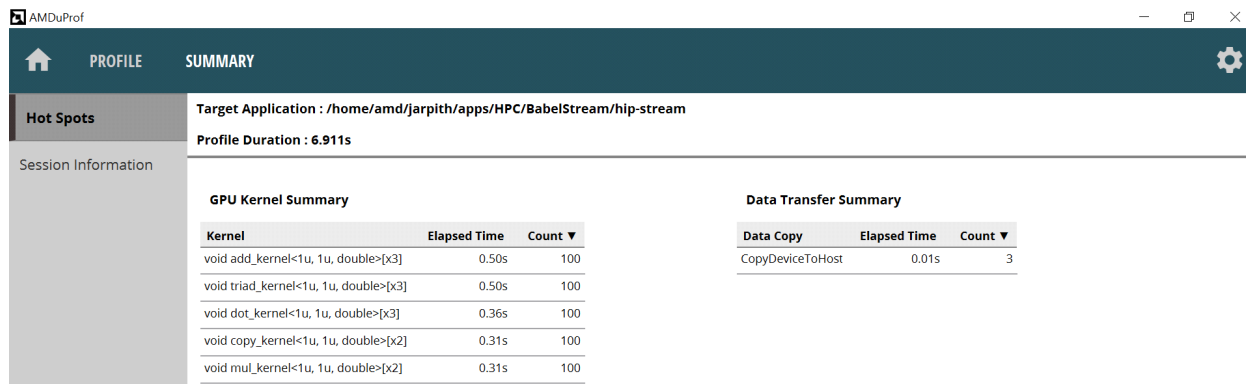
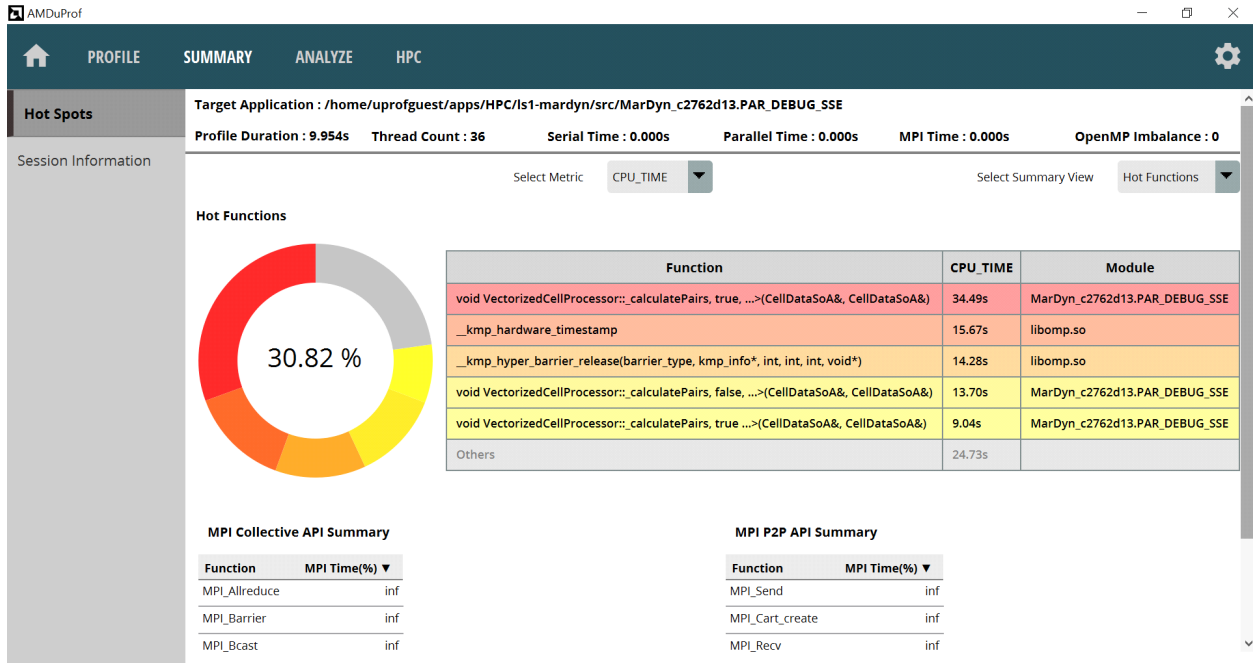


Figure 13. GPU Trace

MPI Trace

The **MPI Trace** screen will look as follows:



CPU Profile

The **CPU Profile** screen will look as Figure 11.

5.5.2 Thread Concurrency Graph

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

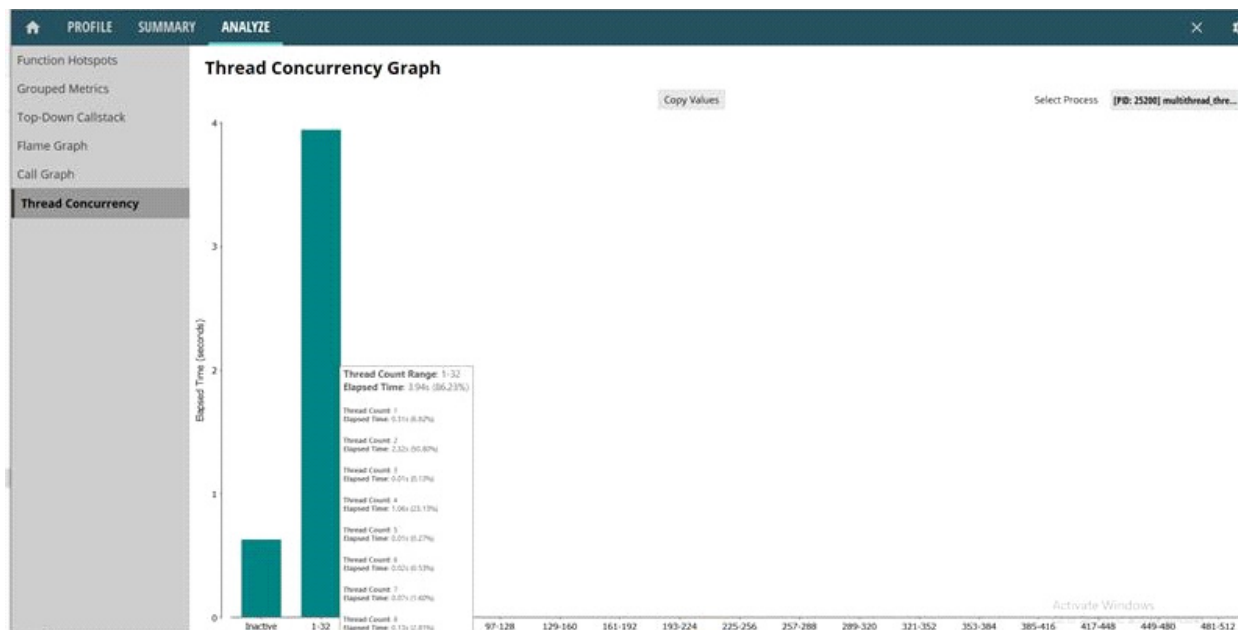


Figure 14. Summary - Thread Concurrency Graph

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

Bucketization approach is used for this graph. Instead of showing the **Elapsed Time** for each core, the weighted average based on the bucket size will be taken. The bucket size will be determined based on the cores and number of available pixels available. This is done to avoid the horizontal scrolling.

5.5.3 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:

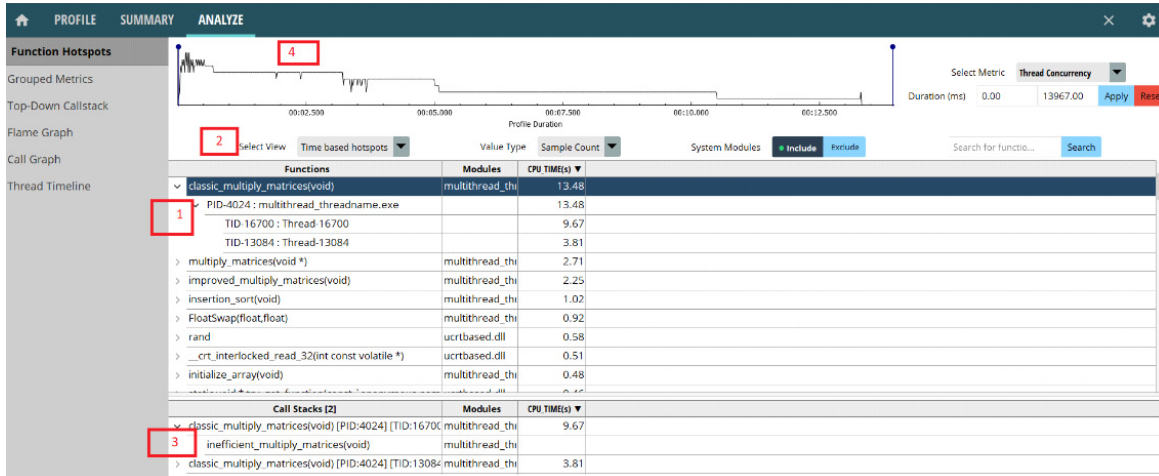


Figure 15. ANALYZE - Function Hotspots

Function Hotspots screen contains the following:

- Process and thread wise breakdown of data is available if the entries are expanded in **Function Hotspots View**. 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.
- 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. When the entries are expanded, process and thread-wise breakdown of data is available.

5.5.4 Process and Functions

Click **ANALYZE > Grouped 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:

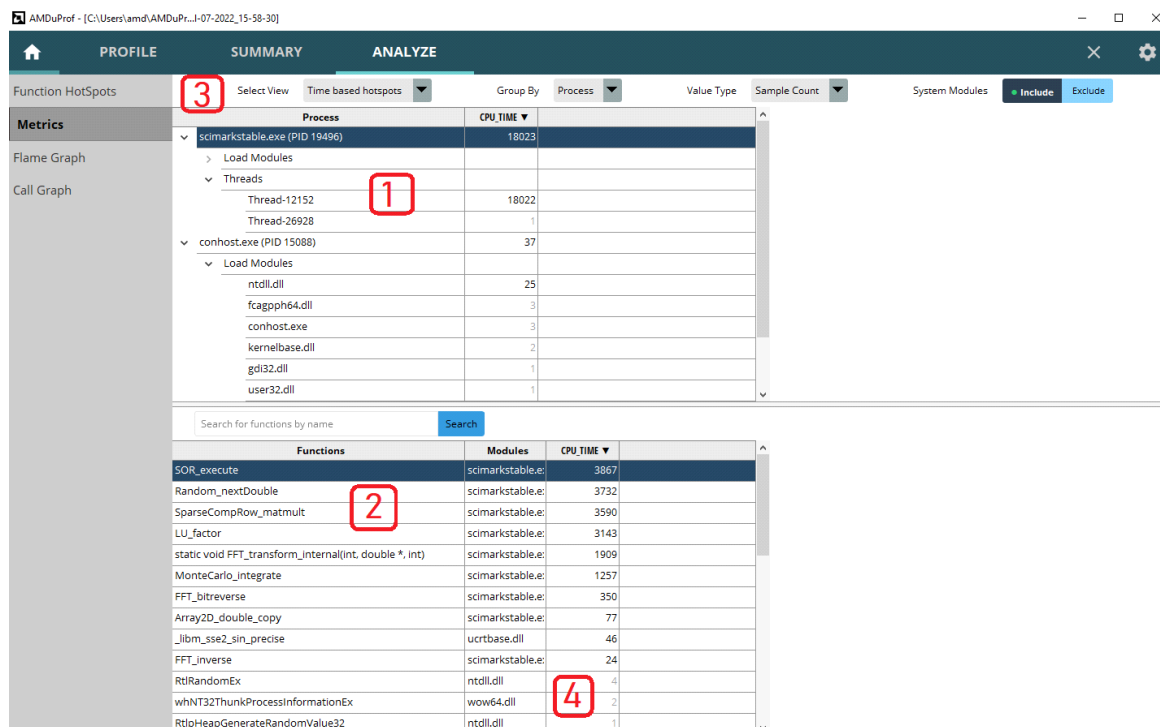


Figure 16. Analyze - Metrics

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.

5.5.5 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:

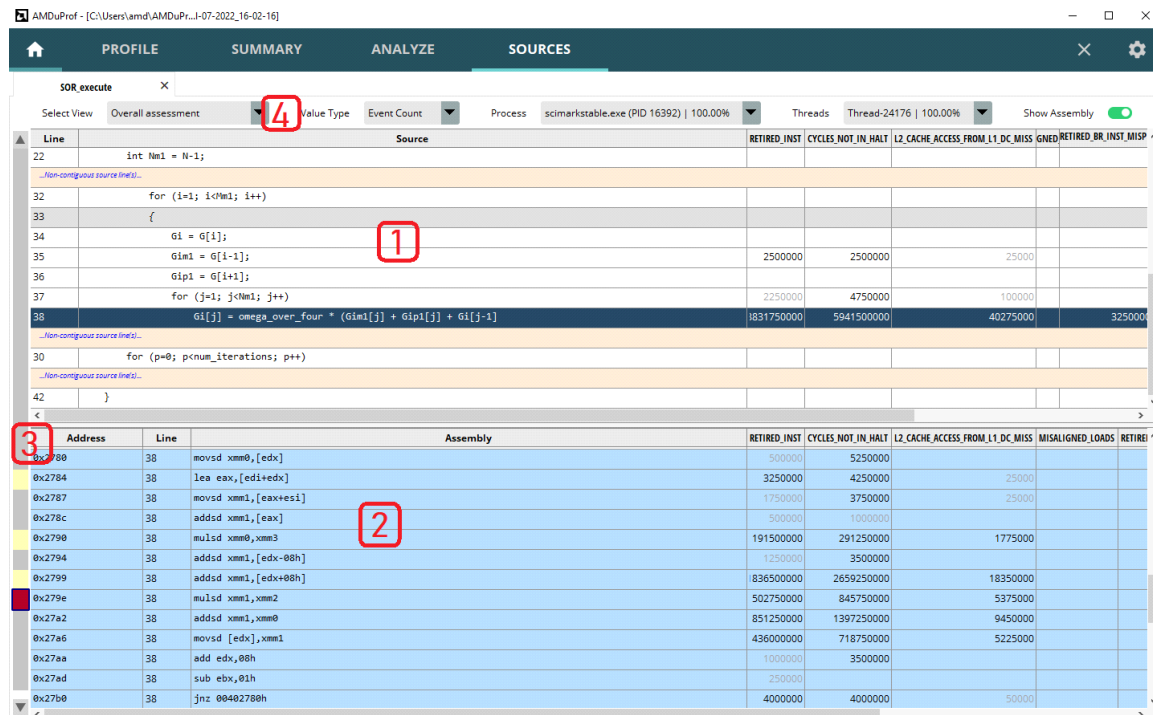


Figure 17. SOURCES - Source and Assembly

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.

5.5.6 Top-down Callstack

Top-down Callstack view can be used to explore the call-sequence flow of the application to analyze the time spent in functions and its callees.

Click **ANALYZE > Top-down Callstack** to view it as follows:

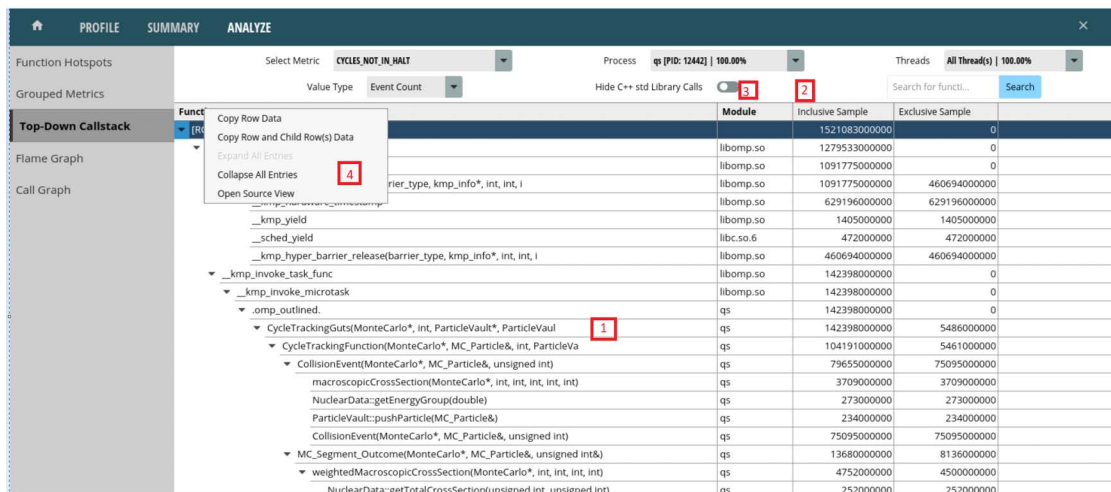


Figure 18. Top-down Callstack

1. Functions are displayed based on the parent to child entries depending on the inclusive samples values sorted.
2. Inclusive sample values for a function and its descendants.
3. Enabling **Hide C++ std Library Calls** option works only when C++ library calls are made. It will exclude such calls from the list and display the other child entries.
4. Context menu of collapse entries will close all the expanded entries. Expand entries will expand the child entries and the **Open Source View** option will display the corresponding source view.

5.5.7 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:

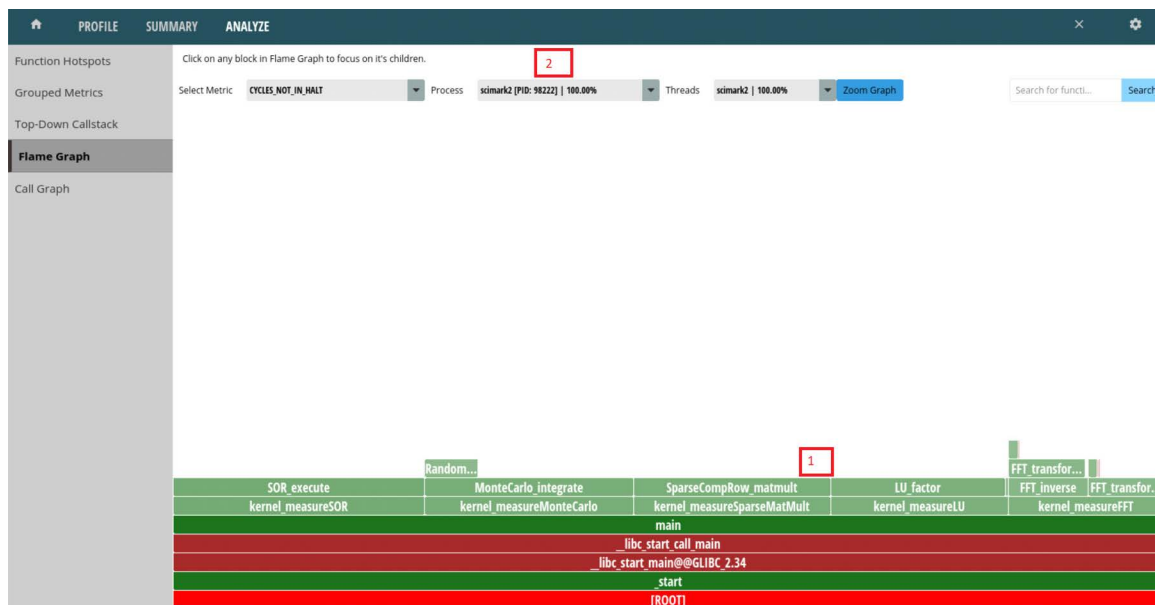


Figure 19. ANALYZE - Flame Graph

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.

5.5.8 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:

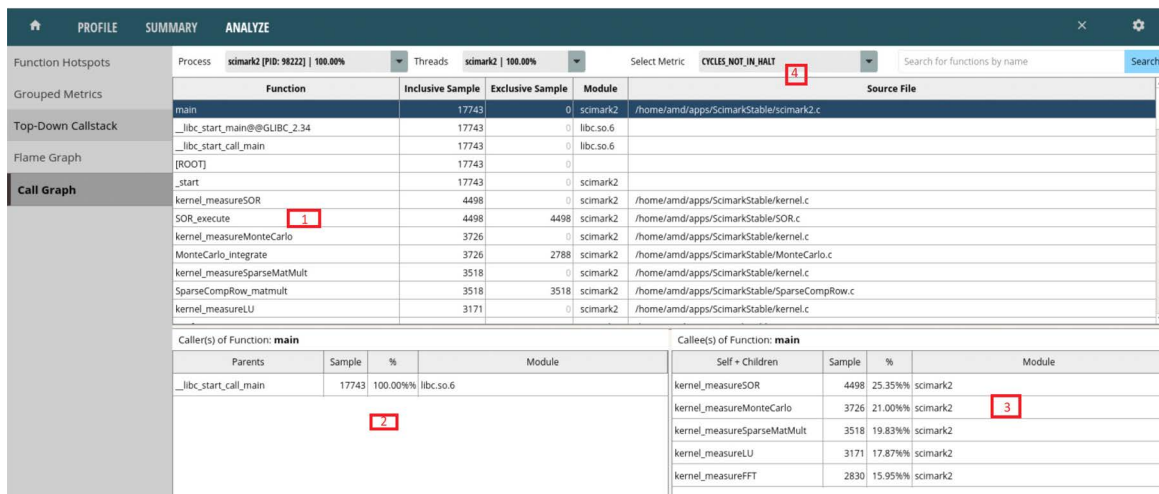


Figure 20. ANALYZE - Call Graph

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.

5.5.9 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:

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
retmq	10	0.00

Figure 21. 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.

5.6 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:

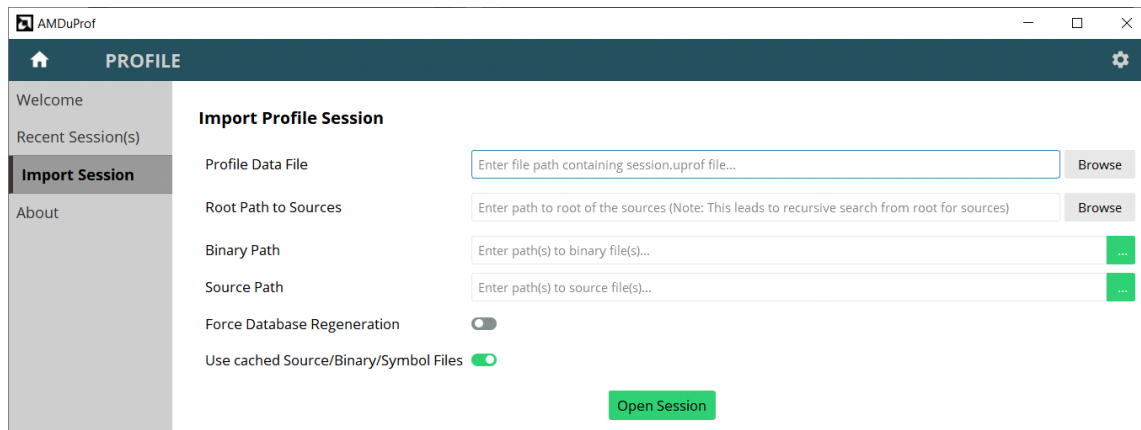


Figure 22. Import Session – Importing Profile Database

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.

5.7 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:

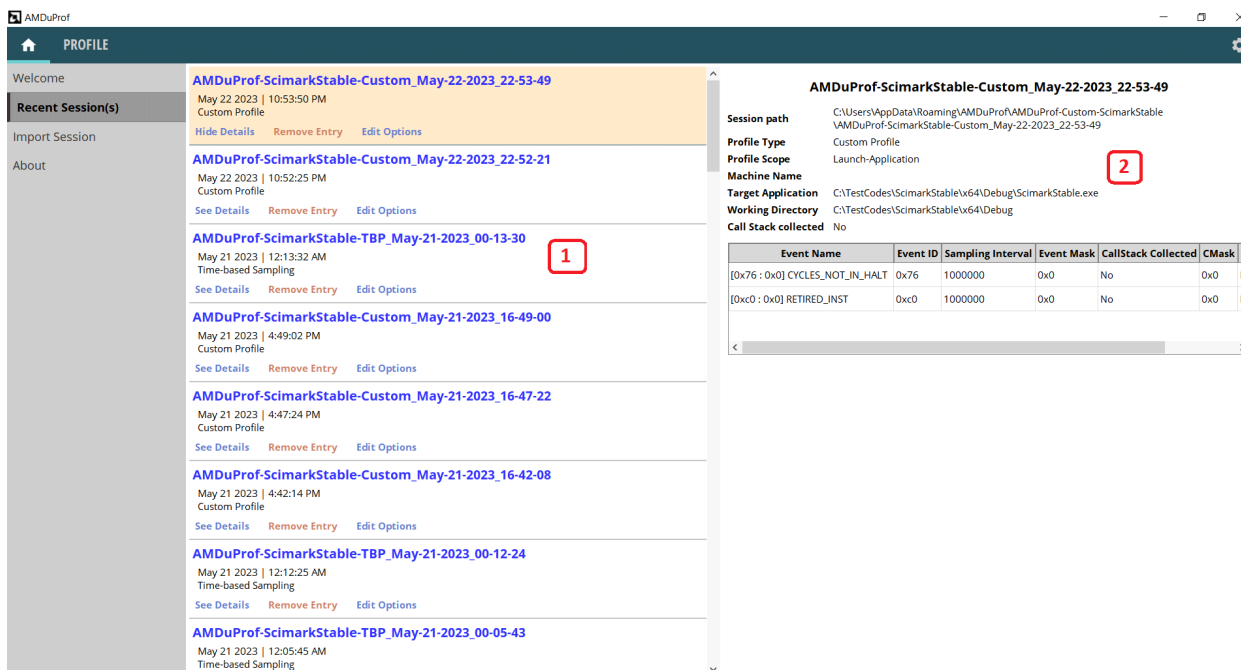


Figure 23. PROFILE - Recent Session(s)

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.

5.8 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:

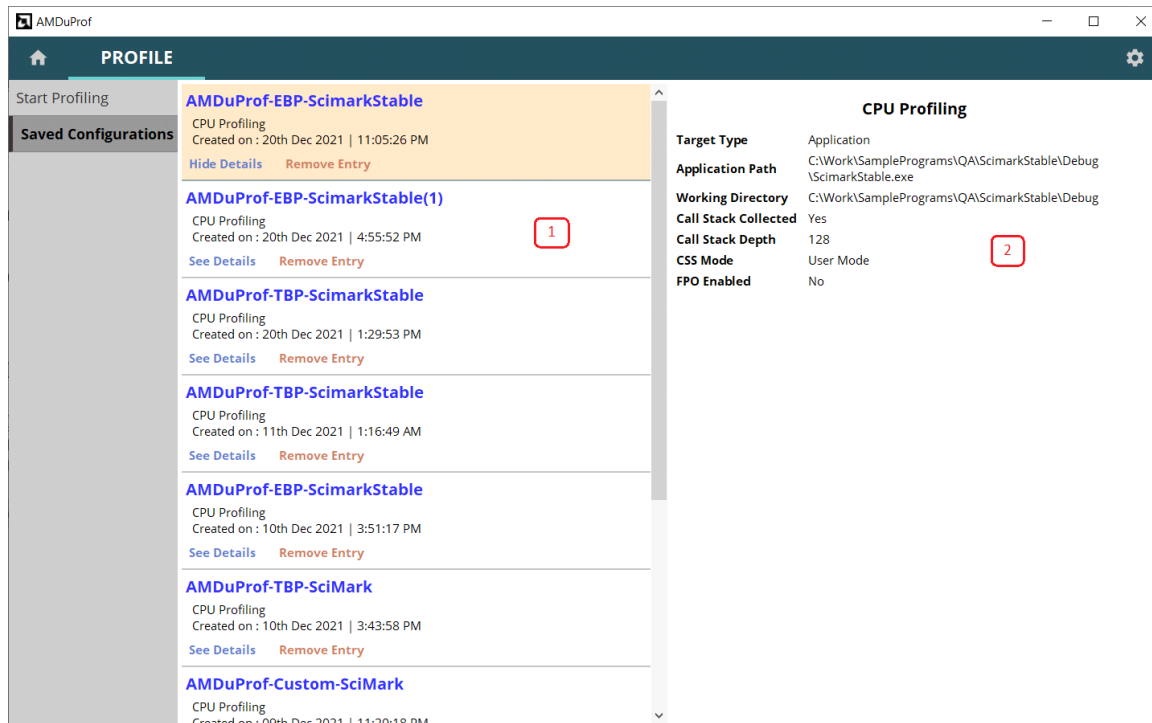


Figure 24. PROFILE - Saved Configurations

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**.

5.9 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.

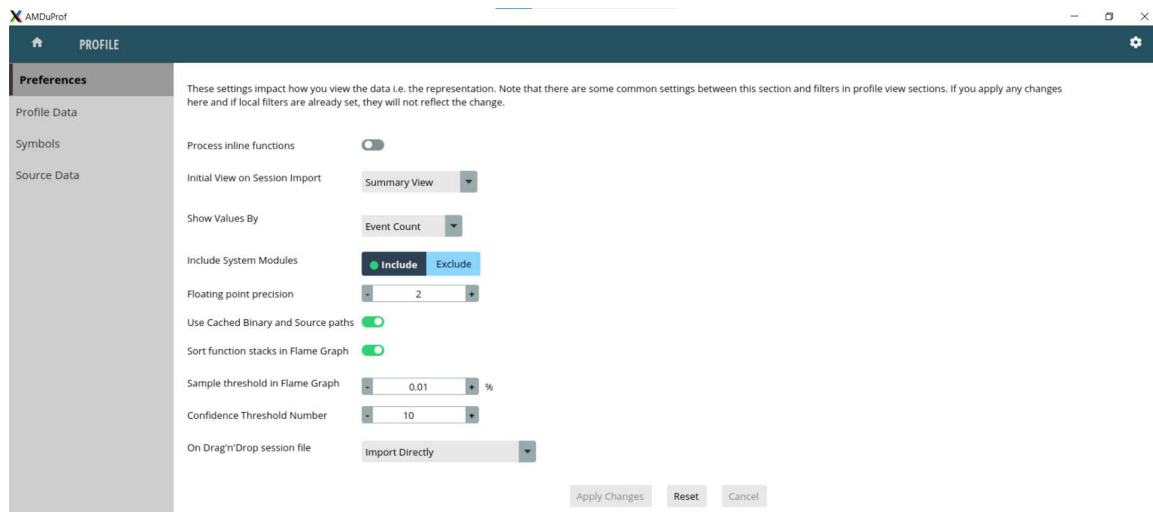


Figure 25. SETTINGS - Preferences

- 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:

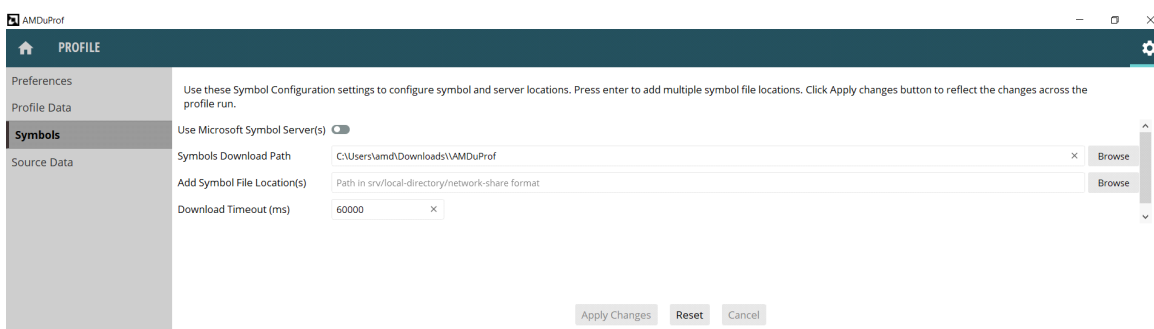


Figure 26. SETTINGS - Symbols

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

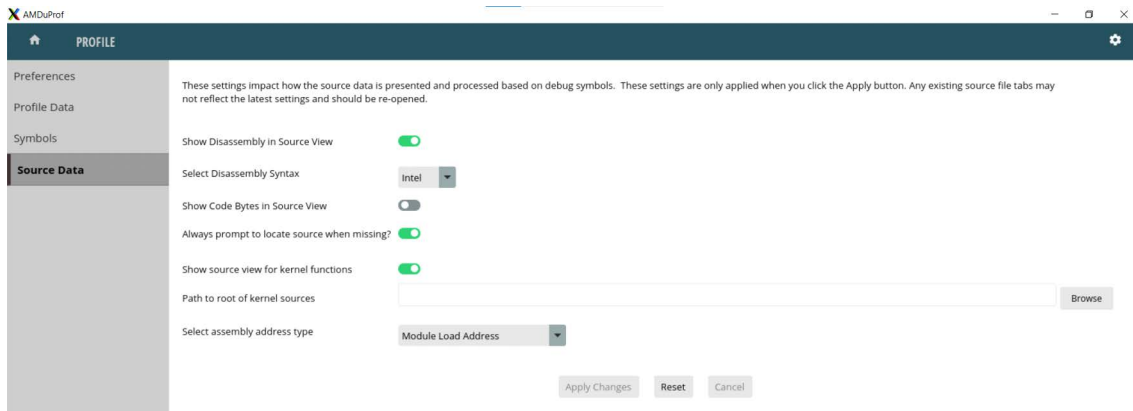


Figure 27. SETTINGS - Source Data

You can use **Select Disassembly Syntax** to select the syntax in which you wish to see the disassembly. By default, it is set to Intel on windows and AT&T on Linux.

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

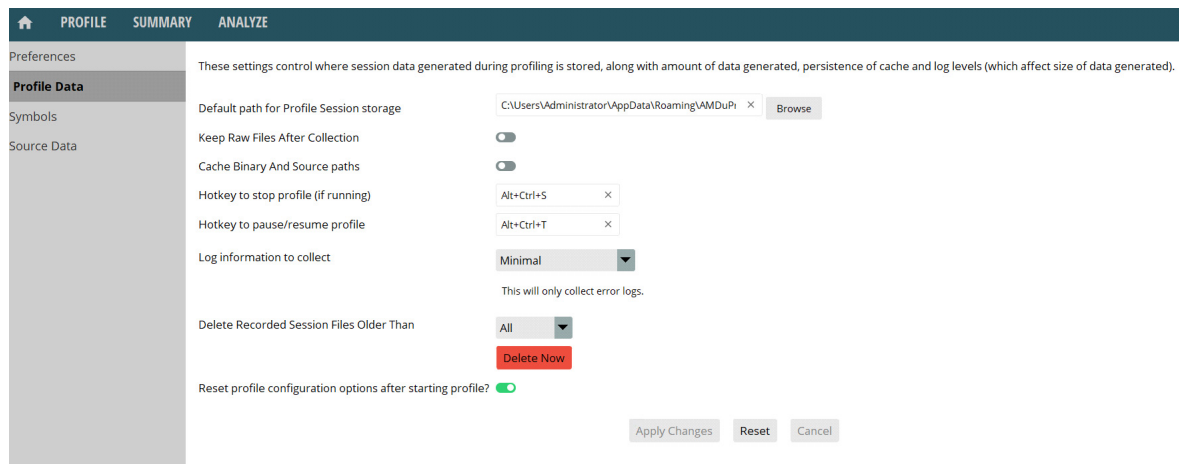


Figure 28. Profile Data

- **Keep Raw Files After Collection** enables saving of the raw files after translation. It is disabled by default.
- You can use the option **Delete Record Session Files** to delete the session files older than a given time period. The time period is set to **None** by default.
- **Reset Profile Configuration** helps add preference to keep or clear the profile configuration after each profile. It is set to **True** (clear after profiling) by default.
- **Hotkey to stop profile (if running)** helps halt the CPU and Power profiling.
- **Hotkey to pause/resume profile** helps pause or resume the CPU and Power profiling

Note: Hotkeys are supported only on Windows.

5.10 Shortcut Keys

Following table lists the AMD uProf shortcut keys:

Table 23. Shortcut Keys

Shortcut Key	Description
Ctrl + O	Import a session.
Ctrl + P	Start configuring a profile, that is, provide an application path to a profile.
Ctrl + S	Jump to the first section of the settings page.
Ctrl + F	Bring focus to the search bar in Function Hotspots. This is applicable for Function Hotspots, Grouped Metrics, Flame Graph, Call Graph, Top-down, and Callstack.
Ctrl + K	Highlight the source line with maximum samples in Source View.
Ctrl +/-	Zoom in/out a timeline view.
Ctrl + Z	Zoom in to a particular region of a timeline view.

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 24. 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.
profile	Collects the performance profile data, analyzes it and generates the profile report.
compare,diff	Processes multiple profile-data and generates a comparison report.

For more information on the workflow, refer to the section “Workflow and Key Concepts”. 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++, and Fortran) application, you must complete the following steps:

1. Prepare the application. For more information on preparing an application for profiling, refer to the section “Reference”.
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 a time-based profile and generate a report for the 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 29. 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:

```
[amd@win-f7f2kcshg7k bin]$ ./AMDuProfCLI 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.

  threading : Threading Analysis
            Use this configuration to get an overall threading analysis and to find
            potential issues for further investigation.
            [PMU Events: PMCx003, PMCx076, PMCx0C0, PMCx043]

  memory   : Cache Analysis
            Use this configuration to identify the false cache-line sharing issues.
            The profile data will be collected using IBS OP.

  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, PMCx28F, PMCx18E, PMCx060, PMCx064, PMCx084, PMCx085,
              PMCx094]

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

  data_access : Investigate Data Access
            Use this configuration to find data access operations with poor L1 data
            cache locality and poor DTLB behavior.
            [PMU Events: PMCx076, PMCx0C0, PMCx029, PMCx060, PMCx043, PMCx047, PMCx045]

  cpi      : Investigate CPI
            Basic profile type to analyse the CPI and IPC metrics of the running application
            or the entire system.
            [PMU Events: PMCx076, PMCx0C0]

  branch   : Investigate Branching
            Use this configuration to find poorly predicted branches and near returns.
            [PMU Events: PMCx076, PMCx0C0, PMCx0C2, PMCx0C3, PMCx0C4, PMCx0C5, PMCx0C8, PMCx0C9,
              PMCx0CA]

  assess_ext : Assess Performance (Extended)
            Use this configuration for an overall assessment of performance and to
            find the potential issues for further investigation. This has additional
            events to monitor than the Assess Performance configuration.
```

Figure 30. Supported Predefined Configurations on Linux

```

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.

  memory   : Cache Analysis
            Use this configuration to identify the false cache-line sharing issues.
            The profile data will be collected using IBS OP.

  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, PMCx28F, PMCx18E, PMCx060, PMCx064, PMCx084, PMCx085,
              PMCx094]

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

  data_access : Investigate Data Access
            Use this configuration to find data access operations with poor L1 data
            cache locality and poor DTLB behavior.
            [PMU Events: PMCx076, PMCx0C0, PMCx029, PMCx060, PMCx043, PMCx047, PMCx045]

  cpi      : Investigate CPI
            Basic profile type to analyse the CPI and IPC metrics of the running application
            or the entire system.
            [PMU Events: PMCx076, PMCx0C0]

  branch   : Investigate Branching
            Use this configuration to find poorly predicted branches and near returns.
            [PMU Events: PMCx076, PMCx0C0, PMCx0C2, PMCx0C3, PMCx0C4, PMCx0C5, PMCx0C8, PMCx0C9,
              PMCx0CA]

  assess_ext : Assess Performance (Extended)
            Use this configuration for an overall assessment of performance and to
            find the potential issues for further investigation. This has additional
            events to monitor than the Assess Performance configuration.
            [PMU Events: PMCx076, PMCx0C0, PMCx0C2, PMCx0C3, PMCx029, PMCx060, PMCx047, PMCx043,
              PMCx024, PMCx052, PMCx00E, PMCx063, PMCx050]

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

```

Figure 31. Supported Predefined Configurations on Windows

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.

- TAKEN BRANCH ANALYSIS SUMMARY — List of the top 10 hot branches
- 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.
 - LAST BRANCH RECORD FOR PROCESS — List of collected branches for the process.
 - 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:

```
[amd@win-f7f2kcsHg7k bin]$ ./AMDuProfCLI timechart --list

Supported Devices:-

Device Name          Instance
-----
Socket               [ 0 - 1 ]
Core                 [ 0 - 255 ]
Thread               [ 0 - 511 ]

Supported Counter Categories:-

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

Figure 32. 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 33. 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 25. 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 25. 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> (value should be in the range 0x0 to 0x7f) • inv=<0 1> • interval=<sampling-interval> • frequency=<frequency (n)> (supported only for Core PMC events, the frequency should be provided in Hz) • ibsop-count-control=<0 1> (for ibs-op event) • loadstore (for ibs-op event, only on Windows platform) • ibsop-l3miss (for IBS OP event, supported only on AMD “Zen4” processors) • ibsfetch-l3miss (for IBS FETCH event, supported only on AMD “Zen4” processors) • call-graph <p><i>Notes:</i></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. • ibsop-l3miss – Enable IBS OP sample collection only when a l3 miss occurs, for example, '-e event=ibs-op,interval=100000,ibsop-l3miss'

Table 25. AMDuProfCLI Collect Command Options

Option	Description
	<ul style="list-style-type: none"> • ibsfetch-l3miss – Enable IBS FETCH sample collection only when a l3 miss occurs, for example, '-e event=ibs-fetch,interval=100000,ibsfetch-l3miss' <p>When these arguments are not passed, then the default values are:</p> <ul style="list-style-type: none"> • umask=0 • cmask=0x0 • 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...>	<p>Profile the existing processes by attaching to a running process. The process IDs are separated by comma.</p> <p><i>Note: A maximum of 512 processes can be attached at a time.</i></p>
-a --system-wide	<p>System Wide Profile (SWP)</p> <p>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.</p>
-c --cpu <core...>	<p>Comma separated list of CPUs to profile. The ranges of CPUs can be specified with '-',for example, 0-3.</p> <p><i>Note: On Windows, the selected cores should belong to only one processor group. For example, 0-63, 64-127, and so on.</i></p>
-d --duration <n>	<p>Profile only for the specified duration n in seconds.</p>
--interval <num>	<p>Sampling interval for PMC events.</p> <p><i>Note: This interval will override the sampling interval specified with individual events.</i></p>
--affinity <core...>	<p>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.</p>
--no-inherit	<p>Do not profile the children of the launched application (processes launched by the profiled application).</p>
-b --terminate	<p>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.</p>
--start-delay <n>	<p>Start delay n in seconds. Start profiling after the specified duration. When n is 0, there is no impact.</p>

Table 25. 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”).
-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 MB) crosses the specified limit. <i>Note: This option may be deprecated in future releases.</i>
--frequency <n> --freq <n> -F <n>	Enable data collection at the specified frequency 'n' (in Hz) for Core PMC events. <i>Note: This frequency will override the sampling frequency specified with the individual events.</i>

6.4.2 Windows Specific Options

The following table lists Linux specific collect commands:

Table 26. 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: If the frame pointers are not available, collect callstack information using unwind information. • fp: Use the frame pointers to collect callstack information.
-g	Same as passing --call-graph 1:128:user:fp.

Table 26. AMDuProfCLI Collect Command – Windows Specific Options

Option	Description
--thread <thread=concurrency>	Collects the runtime thread details
-m --data-buffer-count <size>	Size (number of pages per core) of the buffer used for data collection by the driver. The default size is 512 pages per core.
--trace os	Trace the target domain OS. Support provided for "schedule event" only. Use the command 'info --list trace-events' for a list of OS trace events.
--limit-data <n>	Stop the profiling when the collected data file size (in MB) crosses the specified limit. When used with the option --overwrite, the limit is before the collection is terminated. Size can be specified with the suffix Mega Bytes (M/m), Giga Bytes (G/g), or Seconds (secs).
--overwrite	Specify the profile data collection mode as a ring buffer. The collection limit can be set using the option --limit-data. The default --limit-data is to restrict the raw data file size to 512 pages per core.

6.4.3 Linux Specific Options

The following table lists Linux specific collect commands:

Table 27. 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 dwarf: Collect the process callstack during sample collection and use the DWARF information to reconstruct callstack. fp: Use the frame pointers to collect callstack information. <p>When F = fpo, (N) specifies the max stack-size in bytes to collect per sample collection. Valid range of the stack size: 16 - 32768. 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.

Table 27. AMDuProfCLI Collect Command – Linux Specific Options

Option	Description
--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>,<lwt full>] Provide MPI implementation type: 'openmpi' for tracing OpenMPI library 'mpich' for tracing MPICH and its derivative libraries, for example, Intel MPI Provide tracing scope: 'lwt' for light-weight tracing 'full' for complete tracing '--trace mpi' defaults to '--trace mpi=mpich,full' • 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 <code>info --list trace-events</code> for a list of OS trace events. • user=<event1,event2,...> — provide event names and threshold with comma separated list. These events will be collected in the user mode. Use the command <code>info --list trace-events</code> to get a list of trace events supported in the user mode. • gpu[=<hip,hsa>] — provide the domain for GPU Tracing. By default, the domain is set to 'hip,hsa'.
--buffer-size <size>	<p>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).</p>
--max-threads <thread-count>	<p>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.</p> <ul style="list-style-type: none"> • Launch App - Valid range: 1 to 4096 • System wide - Valid range: 1 to 4194304
--func <module:function-pattern>	<p>Specify functions to trace from the library, executable, or kernel:</p> <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>

Table 27. AMDuProfCLI Collect Command – Linux Specific Options

Option	Description
<pre>--exclude-func <module:function-pattern></pre>	<p>Specify functions to exclude from the library, executable, or kernel:</p> <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>
<pre>-m --mmap-pages <size></pre>	<p>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).</p>
<pre>--mpi</pre>	<p>Pass this option while collecting CPU Profiling data of a MPI application. For MPI tracing, check the --trace option.</p>
<pre>--kvm-guest <pid></pre>	<p>Specify the PID of qemu-kvm process to be profiled to collect guest-side performance profile.</p>
<pre>--guest-kallsyms <path></pre>	<p>Specify the path of guest /proc/kallsyms copied on the local host. AMD uProf reads it to get the guest kernel symbols.</p>
<pre>--guest-modules <path></pre>	<p>Specify the path of guest /proc/modules copied to the local host. AMD uProf reads it to get the guest kernel module information.</p>
<pre>--guest-search-path <path></pre>	<p>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.</p>
<pre>--branch-filter</pre>	<p>Capture LBR data.</p> <p>You can also specify the branch filter type:</p> <ul style="list-style-type: none"> u: user branches k: kernel branches any: any branch type any_call: any call branch any_ret: any return branch ind_call: indirect calls ind_jump: indirect jumps cond: conditional branches call: direct calls <p><i>Notes:</i></p> <ol style="list-style-type: none"> When the above filters not set, the default filter type will be 'any'. This option will work only with PMC events. This is applicable to per process and attach process profiling. However, it is not applicable to Java app profiling.

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
```

- Launch *AMDTClassicMatMul.exe* and collect the samples for PMCx076 and PMCx0C0 events with count-mask enabled:

```
C:\> AMDuProfCLI.exe collect -e event=pmcx076,cmask=0x0, -e
event=pmcx0c0,cmask=0x7f,interval=250000 -o c:\Temp\cpuprof-pmc AMDTClassicMatMul-bin
```

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
```

- Launch *AMDTClassicMatMul-bin* and collect the threading configuration to analyze hotspots, thread state, and wait object analysis among threads:

```
$ AMDuProfCLI collect --config threading -o /tmp/cpuprof-threading 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 all the user mode trace events:

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

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

```
$ 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
```
- Launch *AMDTClassicMatMul-bin* and perform branch analysis with the default filter type:

```
$ AMDuProfCLI collect --branch-filter -o /tmp/cpuprof-ebp-branch AMDTClassicMatMul-bin
```
- Launch *AMDTClassicMatMul-bin* and collect samples for the event PMCXC0:

```
$ AMDuProfCLI collect -e event=pmcxc0,interval=250000 --branch-filter u,k,any -o /tmp/cpuprof-  
ebp-branch AMDTClassicMatMul-bin
```

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 28. 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. <i>Note: A maximum of 512 processes can be attached at a time.</i>
-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. <i>Notes:</i> <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-style <att intel>	Choose the syntax of assembly instructions. The supported options are att and intel. If this option is not used: <ul style="list-style-type: none"> • intel is used by default on Windows. • att is used by default on Linux.
--disasm-only	Generate the function report with only assembly instructions.

Table 28. 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 <low medium high INTERVAL>	<p>Use this option to configure the sample aggregation interval which is useful when the session is imported to GUI.</p> <p>low level of aggregation interval generates better timeline view in GUI but increases the database size.</p> <p>Aggregation INTERVAL can also be specified as a numeric value in milliseconds.</p>
--time-filter <T1:T2>	<p>Restricts report generation to the time interval between T1 and T2. Where, T1 and T2 are time in seconds from profile start time.</p>
--imix	<p>Generate instruction MIX report. It is only supported for IBS config and IBS events profiling. It is only supported for the native binaries.</p>
--ignore-system-module	<p>Ignore samples from system modules.</p>
--show-percentage	<p>Show percentage of samples instead of actual samples.</p>
--show-sample-count	<p>Show the number of samples. This option is enabled by default.</p>
--show-event-count	<p>Show the number of events occurred.</p>
--show-all-cachelines	<p>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.</p>
--bin-path <path>	<p>Binary file path, multiple usage of <code>--bin-path</code> is allowed.</p>
--src-path <path>	<p>Source file path, multiple usage of <code>--src-path</code> is allowed.</p>

Table 28. AMDuProfCLI Report Command Options

Option	Description
--symbol-path <path1;...>	Debug Symbol paths (semicolon separated). Multiple use of --symbol-path is allowed.
--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.
--remove-raw-files	Remove the raw data files to recover the disk space.
--export-session	Create a compressed archive of the required session files which can be used in other system for analysis.
--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 29. 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.
--legacy-symbol-downloader	Download symbols using the Microsoft Symsrv. By default, AMD symbol downloader will be used.

6.5.3 Linux Specific Options

Table 30. AMDuProfCLI Report Command - Linux Specific Options

Option	Description
--host <hostname>	This option is used along with the --input-dir option. Generates report belonging to a specific host. The supported options are: <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>	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: <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. • trace – Generate report specific to trace events. • gputrace – Generate report specific to GPU Tracing. • gpuprof – Generate report specific to GPU Profiling. <p>Example: --category cpu,mpi,trace,gputrace,gpuprof --category mpi --category cpu --category trace --category gputrace --category gpuprof</p>
--funccount-interval <funccount-interval>	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.

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=pmcxc0,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 Trace report from the raw datafile:

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

- Generate GPU Trace report from the raw datafile:

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

- 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 31. AMDuProfCLI Translate Command Options

Option	Description
-h --help	Displays the help information.
-i --input-dir <directory-path>	Path to the directory containing collected data.
--time-filter <T1:T2>	Restricts the processing to the time interval between T1 and T2, where T1, T2 are time in seconds from profile start time.
--agg-interval <low medium high INTERVAL>	Use this option to configure the sample aggregation interval which is useful when the session is imported to GUI. low level of aggregation interval generates better timeline view in GUI but increases the database size. Aggregation INTERVAL can also be specified as a numeric value in milliseconds.
--bin-path <path>	Binary file path. Multiple use of --bin-path is allowed.
--symbol-path <path>	Debug symbol path. Multiple instances of --symbol-path are allowed.
--inline	Inline function extraction for C and C++ executables. Notes: 1. This option is not supported on Windows. 2. Using this option will increase the time taken to generate the report.
--retranslate	Re-translate the 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 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 the --enable-log option.
--remove-raw-files	Remove the raw data files to recover the disk space.
--export-session	Create a compressed archive of required session files which can be used in other system for analysis.

6.6.2 Windows Specific Options

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

Table 32. 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.
<code>--legacy-symbol-downloader</code>	Download symbols using the Microsoft Symsrv. By default, AMD symbol downloader will be used.

6.6.3 Linux Specific Options

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

Table 33. 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>trace</code> - User mode tracing • <code>gputrace</code> - GPU Tracing • <code>gpuprof</code> - GPU Profiling Example: <code>--category cpu,mpi,trace,gputrace,gpuprof</code> <code>--category mpi --category cpu --category trace --category gputrace --category gpuprof</code>
<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: <code><hostname></code> : Translate only the processes belonging to a specific host. <code>all</code> : Translate all processes <i>Note: 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>
<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 trace raw data file:

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

- 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 34. 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: <ul style="list-style-type: none"> • 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 Diff Command

The `diff` command streamlines the process of comparing multiple profile reports by automating the manual comparison of events. It processes the raw profile data, processed files, or database files to generate a markdown comparison report for the collected profiles. The generated markdown file includes detailed function data providing comprehensive insights into the compared profiles.

Furthermore, the `diff` command can also be used to generate a single profile report by specifying only the base profile path. This simplifies the generation of individual reports, making it more convenient and efficient.

During profile comparison, there is always a single base profile and multiple non-base profiles. Valid comparison results are obtained only for the functions that exist in both the base profile and non-base profiles.

By default, the comparison results are displayed in the source view. In the source view table, information, such as File, Line, Source Code, Address, Instruction, Code Byte, and Events are provided for each function. This comprehensive view enables a detailed analysis of the compared profiles.

Note: *To obtain meaningful and accurate comparison results, it is important to ensure that the base profile and non-base profiles have matching functions available for comparison.*

Synopsis:

```
AMDuProfCLI diff [--help] [<options>]
AMDuProfCLI compare [--help] [<options>]
```

Common Usages:

```
AMDuProfCLI diff --baseline <base session-dir path> --with <non-base session-dir path> -o
<output-dir>
```

6.8.1 Profile Comparison Eligibility Criteria

To ensure accurate and meaningful profile comparisons, the following conditions must be met:

- **Same Events:** The profiles being compared should have collected the same events. This ensures that the comparison is performed on relevant and comparable data.
- **Same Profile Duration (if specified):** If the duration (`-d`) option is specified, the profiles being compared should have the same duration. This ensures consistency in the time span covered by the profiles.
- **Not a System Wide Profile:** System-wide profiles cannot be compared directly. Therefore, only individual process or thread-level profiles are eligible for comparison.
- **Same Profile Data Limit (if used):** If the `--limit-size` or `--limit-data` option is used during profiling, the profiles being compared should have the same data limit set. This ensures consistency in the size of profile data collected.
- **Same Inline Function Profiling (`--inline`):** If the `--inline` option is used to profile inline functions, the profiles being compared should have used the same inline function profiling setting. This ensures consistent handling of inline functions during the comparison.

6.8.2 Options

Following table lists the `diff` commands:

Table 35. AMDuProfCLI diff Command Options

Option	Description
<code>-h --help</code>	Displays this help information on the console/terminal.
<code>--baseline <directory-path></code>	Path to the directory containing collected data. The profile data in this directory will be treated as the base profile against which all other profiles will be compared.
<code>--with <directory-path></code>	Path to the directory containing collected data. Each profile specified with <code>--with</code> will be considered as a non-base profile and compared against the base profile. You can use multiple instances of <code>--with</code> to specify multiple non-base profiles for comparison.
<code>-i, --input-dir <directory-path></code>	Path to the directory containing collected data. Multiple occurrences of <code>-i</code> is allowed. First occurrence of <code>-i</code> is considered as the base session, while all the subsequent occurrences of <code>-i</code> are treated as non-base sessions. <i>Note: When using <code>-i, --input-dir</code>, you should not use the <code>--baseline</code> or <code>--with</code> options in conjunction. If you use <code>--baseline</code> and <code>-i</code> together, the <code>--baseline</code> option will take precedence and be considered as the base session. If the <code>--baseline</code> option is not present, the first occurrence of <code>-i</code> will automatically be considered as the base session.</i>
<code>--output-dir -o <directory-path></code>	Path where the markdown comparison report will be generated.

Table 35. AMDuProfCLI diff Command Options

Option	Description
--type <comparison-type>	<p>Specify the type of comparison to be performed. The supported comparison types are:</p> <ul style="list-style-type: none"> • name: With this type, only the top ‘n’ functions from the base profile will be compared with the corresponding functions available in the non-base profiles. The comparison will focus on the similar functions between the profiles. • order: With this type, the top ‘n’ functions from all the profiles will be displayed in the order of profiles. The order will be: base profile first, followed by the 1st non-base profile, 2nd non-base profile, and so on. The comparison will still be performed with the functions present in the base profile and only for the similar functions across the profiles. <p>The default comparison type is name.</p>
--alias <base-fun,non-base-fun,... base-fun-1,non-base-fun-1,... ...>	<p>In the cases where the function names have changed in the non-base profile, specify the function names in the non-base profile that should be compared with the corresponding function names in the base profile.</p> <p>Specify different functions using the pipe symbol ‘ ’ as a separator. For each set of functions, you can use a comma to separate the function names between the base profile and the non-base profile.</p>
--show-percentage	Comparison results will be displayed in terms of percentages.
--cutoff <n>	Cut-off to limit the number of functions to be reported. ‘n’ is the maximum number of entries to be reported in various report sections. The default value is 10.
--sort-by -s <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> • user=<0 1> • os=<0 1> <p>Use the command <code>info --list pmu-events</code> for the list of supported PMC events. The arguments details:</p> <ul style="list-style-type: none"> • umask — Unit mask in decimal or hexadecimal. Applicable only to the PMC events. • user, os — User and OS mode. Applicable only to the PMC events. <p>Multiple occurrences of <code>-sort-by (-s)</code> are not allowed.</p>
--view <view-config>	Compare 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.
--stdout	Comparison report will also be displayed in the terminal or command line interface apart from saving to a file.

Table 35. AMDuProfCLI diff Command Options

Option	Description
<code>--src-path <path1;...></code>	Source file directories (semicolon separated paths) for base profile. This will be considered for the non-base profiles if the corresponding file directories are not specified separately. Multiple use of <code>--src-path</code> is allowed.
<code>--bin-path <path1;...></code>	Binary file path for the base profile. This will be considered for the non-base profiles if the corresponding bin path is not specified separately. Multiple usage of <code>--bin-path</code> is allowed.
<code>--src-path1 <path1;...></code>	Source file directories (semicolon separated paths) for the 1 st non-base profile. Multiple use of <code>--src-path1</code> is allowed.
<code>--bin-path1 <path1;...></code>	Binary file path for the 1 st non-base profile. Multiple usage of <code>--bin-path1</code> is allowed.
<code>--src-path2 <path1;...></code>	Source file directories (semicolon separated paths) for the 2 nd non-base profile. Multiple use of <code>--src-path2</code> is allowed.
<code>--bin-path2 <path1;...></code>	Binary file path for the 2 nd non-base profile. Multiple usage of <code>--bin-path2</code> is allowed.
<code>--src-path3 <path1;...></code>	Source file directories (semicolon separated paths) for the 3 rd non-base profile. Multiple use of <code>--src-path3</code> is allowed.
<code>--bin-path3 <path1;...></code>	Binary file path for the 3 rd non-base profile. Multiple usage of <code>--bin-path3</code> is allowed.

6.8.3 Examples

Windows

Use the following commands to:

- Generate a comparison report of base profile data with subsequent profile data:

```
C:\> AMDuProfCLI.exe diff --baseline c:\Temp\cpuprof-tbp\<BASE-DIR> --with c:\Temp\cpuprof-tbp\<NON-BASE-DIR> -o c:\Temp\cpuprof-tbp
```

- Generate a comparison report using the `-i` option:

```
C:\> AMDuProfCLI.exe diff -i c:\Temp\cpuprof-tbp\<BASE-DIR> -i c:\Temp\cpuprof-tbp\<NON-BASE-DIR> -o c:\Temp\cpuprof-tbp
```

- Generate a comparison report without ignoring the unique entries across sessions:

```
C:\> AMDuProfCLI.exe diff --baseline c:\Temp\cpuprof-tbp\<BASE-DIR> --with c:\Temp\cpuprof-tbp\<NON-BASE-DIR> --type order -o c:\Temp\cpuprof-tbp
```

- Generate a comparison report of base profile data with subsequent profile data sorted on ibs-op event:

```
C:\> AMDuProfCLI.exe diff --baseline c:\Temp\cpuprof-tbp\<BASE-DIR> --with c:\Temp\cpuprof-tbp\<NON-BASE-DIR> --type name -s ibs-op -o c:\Temp\cpuprof-tbp
```

- Generate a comparison report with delta shown in percentage:

```
C:\> AMDuProfCLI.exe compare --baseline c:\Temp\cpuprof-tbp\<BASE-DIR> --with c:\Temp\cpuprof-tbp\<NON-BASE-DIR> --type name --show-percentage -o c:\Temp\cpuprof-tbp
```

- Generate a comparison report of base profile data with successor profile data with changed function names across sessions:

```
C:\> AMDuProfCLI.exe compare --baseline c:\Temp\cpuprof-tbp\<BASE-DIR> --with c:\Temp\cpuprof-tbp\<NON-BASE-DIR> --alias CalculateSum,CalculateUpdatedSum|enhanceOutput,optimizeOutput -o c:\Temp\cpuprof-tbp
```

- Generate a comparison report of base profile data with multiple successor profile data:

```
C:\> AMDuProfCLI.exe diff -i c:\Temp\cpuprof-tbp\<BASE-DIR> -i c:\Temp\cpuprof-tbp\<NON-BASE-DIR1> -i c:\Temp\cpuprof-tbp\<NON-BASE-DIR2> --with c:\Temp\cpuprof-tbp\<NON-BASE-DIR3> -o c:\Temp\cpuprof-tbp
```

- Generate a comparison report on one of the predefined views:

```
C:\> AMDuProfCLI.exe diff -i c:\Temp\cpuprof-tbp\<BASE-DIR> -i c:\Temp\cpuprof-tbp\<NON-BASE-DIR> --view ipc_assess -o c:\Temp\cpuprof-tbp
```

- Generate a comparison report providing the source and binary paths:

```
C:\> AMDuProfCLI.exe diff -i c:\Temp\cpuprof-tbp\<BASE-DIR> -i c:\Temp\cpuprof-tbp\<NON-BASE-DIR> --bin-path Examples\AMDTClassicMatMul\bin\ --src-path Examples\AMDTClassicMatMul\ --bin-path1 Examples\AMDTClassicMatMulMod\bin\ --src-path1 Examples\AMDTClassicMatMulMod\ -o c:\Temp\cpuprof-tbp
```

Linux

- Generate a comparison report of base profile data with subsequent profile data:

```
$ AMDuProfCLI diff --baseline /tmp/cpuprof-tbp/<BASE-DIR> --with /tmp/cpuprof-tbp/<NON-BASE-DIR> -o /tmp/cpuprof-tbp
```

- Generate a comparison report of base profile data with subsequent profile data sorted on PMC event:

```
$ AMDuProfCLI diff --baseline /tmp/cpuprof-tbp/<BASE-DIR> --with /tmp/cpuprof-tbp/<NON-BASE-DIR> -s event=pmcxc0,user=1,os=0 -o /tmp/cpuprof-tbp
```


6.9 Profile Command

The `profile` command collects the performance profile data, processes it, and generates a profile report in a readable format. It is an alternative to the combination of `collect` and `report` command.

Synopsis:

```
AMDuProfCLI profile [--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 profile <PROGRAM> [<ARGS>]
```

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

6.9.1 Options

Following table lists the `profile` commands:

Table 36. AMDuProfCLI profile Command Options

Option	Description
-h --help	Displays the help information on the console/terminal.
-o --output-dir <directory-path>	Base directory path in which the 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 36. AMDuProfCLI profile Command Options

Option	Description
-e --event or <predefined-event>	<p>A predefined event can 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> • mask=<unit-mask> • user=<0 1> • os=<0 1> • cmask=<count-mask> (Value should be in the range 0x0 to 0x7f) • inv=<0 1> • interval=<sampling-interval> • frequency=<frequency (n)> (Supported only for Core PMC events. Frequency should be provided in Hz) • ibsop-count-control=<0 1> (for ibs-op event) • loadstore (for ibs-op event, only on Windows platform) • ibsop-l3miss (for ibs-op event, supported only on AMD “Zen4” processors) • ibsfetch-l3miss (for ibs-fetch event, supported only on AMD “Zen4” processors) • call-graph <p>Notes:</p> <ol style="list-style-type: none"> 1. Providing unmask with predefined event is not required 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. • ibsop-l3miss – Enable IBS OP sample collection only when a l3 miss occurs, for example, '-e event=ibs-op,interval=100000,ibsop-l3miss' • ibsfetch-l3miss – Enable IBS FETCH sample collection only when a l3 miss occurs, for example, '-e event=ibs-fetch,interval=100000,ibsfetch-l3miss'

Table 36. AMDuProfCLI profile 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=0x0 • user=1 • os=1 • inv=0 • ibsop-count-control=0 (for ibs-op 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...>	<p>Profile the existing processes by attaching to a running process. The process IDs are separated by comma.</p> <p><i>Note: A maximum of 512 processes can be attached at a time.</i></p>
-a --system-wide	<p>System Wide Profile (SWP)</p> <p>If this flag is not set, the command line tool will profile only the launched application or the Process IDs attached with -p option.</p>
-c --cpu <core...>	<p>Comma separated list of CPUs to profile. The ranges of CPUs can be specified with '-', for example, 0-3.</p> <p><i>Note: On Windows, the selected cores should belong to only one processor group. For example, 0-63, 64-127, and so on.</i></p>
-d --duration <n>	<p>Profile only for the specified duration 'n' in seconds.</p>
--interval <num>	<p>Sampling interval for the PMC events.</p> <p><i>Note: This interval will override the sampling interval specified with individual events.</i></p>
--affinity <core-id...>	<p>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.</p>
--no-inherit	<p>Do not profile the children of the launched application (processes launched by the profiled application).</p>
-b --terminate	<p>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.</p>
--thread <thread=concurrency>	<p>Thread concurrency</p>
--start-delay <n>	<p>Start delay n in seconds. Start profiling after the specified duration. When 'n' is 0, there is no impact.</p>

Table 36. AMDuProfCLI profile Command Options

Option	Description
<code>--start-paused</code>	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” section).
<code>-w --working-dir <path></code>	Specify the working directory. The default is the current working directory.
<code>--log-path <path-to-logdir></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 path set by <code>AMDUPROF_LOGDIR</code> environment variable or <code>\$TEMP</code> path (Linux, FreeBSD) or <code>%TEMP%</code> path (on Windows) 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. It should be used with <code>--enable-log</code> option.
<code>--limit-size <n></code>	Use this option to stop the profiling once the collected data file size (in MBs) crosses the limit. This option will be deprecated in future releases.
<code>--frequency <n> --freq <n> -F <n></code>	Enable data collection at the specified frequency 'n' (in Hz) for Core PMC events. <i>Note: This frequency will override the sampling frequency specified with individual events.</i>
<code>--detail</code>	Generate detailed report.
<code>--group-by <section></code>	Specify the report to be generated. The supported report options are: <ul style="list-style-type: none"> • <code>process</code>: Report process details • <code>module</code>: Report module details • <code>thread</code>: Report thread details This option is applicable only with the <code>--detail</code> option. The default is <code>group-by process</code> .
<code>--cutoff <n></code>	Cut-off 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.
<code>--view <view-config></code>	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.
<code>--inline</code>	Show inline functions for C, C++ executables. <i>Notes:</i> <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.
<code>--show-sys-src</code>	Generate detailed function report of the system module functions (if debug info is available) with the source statements.

Table 36. AMDuProfCLI profile Command Options

Option	Description
<code>--src-path <path1;...></code>	Source file directories (semicolon separated paths). Multiple use of <code>--src-path</code> is allowed.
<code>--disasm</code>	Generate a detailed function report with assembly instructions.
<code>--disasm-only</code>	Generate the function report with only assembly instructions.
<code>--disasm-style <att intel></code>	Choose the syntax of assembly instructions. Supported options are 'att' or 'intel'. If this option is not used, the default style used is 'intel'.
<code>-s --sort-by <EVENT></code>	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: <ul style="list-style-type: none"> • <code>event=<timer ibs-fetch ibs-op pmcxNNN></code>, where NNN is hexadecimal Core PMC event id. • <code>umask=<unit-mask></code> • <code>cmask=<count-mask></code> • <code>inv=<0 1></code> • <code>user=<0 1></code> • <code>os=<0 1></code> Use the command <code>info --list pmu-events</code> for the list of supported PMC events. Argument details are: <ul style="list-style-type: none"> • <code>umask</code> — Unit mask in decimal or hexadecimal, applicable only to the PMC events. • <code>cmask</code> — Count mask in decimal or hexadecimal, applicable only to the PMC events. • <code>user, os</code> — User and OS mode. Applicable only to the PMC events. • <code>inv</code> — Invert Count Mask, applicable only to the PMC events • Multiple occurrences of <code>-sort-by (-s)</code> are not allowed.
<code>--agg-interval <low medium high INTERVAL></code>	Use this option to configure the sample aggregation interval which is useful when the session gets imported to GUI. 'low' level of aggregation interval generates better timeline view in GUI, but increases the database size. Aggregation INTERVAL can also be specified as numeric value in milliseconds.
<code>--time-filter <T1:T2></code>	Restricts report generation to the time interval between T1 and T2. Where, T1 and T2 are time in seconds from profile start time.
<code>--imix</code>	Generate the instruction MIX report. It is only supported for IBS config, IBS events profiling, and the native binaries.
<code>--ignore-system-module</code>	Ignore samples from system modules.
<code>--show-percentage</code>	Show percentage of samples instead of actual samples.

Table 36. AMDuProfCLI profile Command Options

Option	Description
--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 --bin-path is allowed.
--src-path <path>	Source file path, multiple usage of --src-path is allowed.
--symbol-path <path1;...>	Debug Symbol paths (semicolon separated). Multiple use of --symbol-path is allowed.
--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, 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.
--ascii event-dump	Use this option to generate ASCII dump of IBS OP profile samples. <i>Note: This option might delay the translation.</i>
--no-report	Use this option to perform only collection and translation.
--remove-raw-files	Removes the raw data files to reclaim the disk space.
--export-session	Use this option to create a compressed archive of required session files which can be used in other system for analysis.

6.9.2 Windows Specific Options

Following table lists Windows specific profile commands:

Table 37. AMDuProfCLI Windows profile Command Options

Option	Description
<code>--call-graph <I:D:S:F></code>	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: If frame pointers are not available, collect callstack information using unwind information. • fp: Use frame pointers to collect callstack information.
<code>-g</code>	Same as passing <code>--call-graph 1:128:user:fp</code> .
<code>--thread <thread=concurrency></code>	Collects the runtime thread details.
<code>-m --data-buffer-count <size></code>	Size (number of pages per core) of the buffer used for data collection by the driver. The default size is 512 pages per core.
<code>--trace os</code>	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.
<code>--symbol-server <path1;...></code>	Symbol Server directories (semicolon separated paths). For example, Microsoft Symbol Server (https://msdl.microsoft.com/download/symbols). Multiple use of <code>--symbol-server</code> is allowed.
<code>--symbol-cache-dir <path></code>	The path to store the symbol files downloaded from the Symbol Servers.
<code>--legacy-symbol-downloader</code>	Use this option to download symbols using the Microsoft Symstrv. By default AMD symbol downloader will be used to download symbols
<code>--limit-data <n></code>	Use this option to stop the profiling once the collected data file size (in MBs) crosses the limit. When used with (<code>--overwrite</code>) option, the limit is before the collection is terminated. Size can be specified with a suffix Mega bytes (M/m), Giga Bytes (G/g), and Seconds (secs).
<code>--overwrite</code>	Specify the profile-data collection mode as a ring buffer. Collection limit can be set using <code>--limit-data</code> option. Default <code>--limit-data</code> is to restrict the raw data file size to 512 pages per core.

6.9.3 Linux Specific Options

Following table lists the Linux specific commands:

Table 38. AMDuProfCLI Linux profile Command Options

Option	Description
<code>--call-graph <F:N></code>	<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"> • <code>fpo dwarf</code>: Collect process call stack during sample collection and use DWARF information to reconstruct the call stack. • <code>fp</code>: Use Frame pointers to collect call stack information. <p>When <code>F = fpo</code>, (N) specifies the max stack-size in bytes to collect per sample collection. Valid range of the stack size: 16 - 32768. If (N) is not a 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 <code>F = fp</code>, the value for N is ignored and hence, there is no need to pass it.</p>
<code>-g</code>	Same as passing <code>--call-graph fp</code>
<code>--tid <TID,...></code>	Profile existing threads by attaching to a running thread. The thread IDs are separated by comma.
<code>--trace <TARGET></code>	<p>To trace a target domain. TARGET can be one or more of the following: <code>mpi[=<openmpi mpich>,<lwt full>]</code> Provide MPI implementation type: 'openmpi' for tracing OpenMPI library 'mpich' for tracing MPICH and its derivative libraries, for example, Intel MPI Provide tracing scope: 'lwt' for light-weight tracing 'full' for complete tracing '<code>--trace mpi</code>' defaults to '<code>--trace mpi=mpich,full</code>'</p> <ul style="list-style-type: none"> • <code>openmp</code> — for tracing OpenMP application. This is same as the option <code>--omp</code>. • <code>os[=<event1,event2,...>]</code> — provide the event names and optional threshold with a comma separated list. <code>syscall</code> and <code>memtrace</code> events will take the optional threshold value as <code><event:threshold></code>. Use the command <code>info --list ostrace-events</code> for a list of the OS trace events. • <code>user=<event1,event2,...></code> — provide the event name and threshold with a comma separated list. These events will be collected in the user mode. Use the command <code>info --list trace-events</code> to get a list of the trace events supported in user mode. • <code>gpu[=<hip,hsa>]</code> — provide the domain for GPU Tracing. By default, the domain is set to 'hip,hsa'.

Table 38. AMDuProfCLI Linux profile Command Options

Option	Description
<code>--buffer-size <size></code>	Number of pages to be allotted for OS trace buffer. The default value is 256 pages per core. Increase the pages to reduce the trace data loss. This option is only applicable to OS tracing (<code>--trace os</code>).
<code>--max-threads <thread-count></code>	Maximum number of threads for OS tracing. The default value is 1024 for launched application and 32768 for System Wide Tracing (<code>-a</code> 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
<code>--func <module:function-pattern></code>	Specify functions to trace from the library, executable, or kernel: 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'. <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>
<code>--exclude-func <module:function-pattern></code>	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'. <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>
<code>-m --mmap-pages <size></code>	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).
<code>--mpi</code>	Pass this option while collecting CPU Profiling data of a MPI application. For MPI tracing, check the <code>--trace</code> option.
<code>--kvm-guest <pid></code>	Specify the PID of qemu-kvm process to be profiled to collect guest-side performance profile.
<code>--guest-kallsyms <path></code>	Specify the path of guest <code>/proc/kallsyms</code> copied on the local host. AMD uProf reads it to get the guest kernel symbols.
<code>--guest-modules <path></code>	Specify the path of <code>guest/proc/modules</code> copied to the local host. AMD uProf reads it to get the guest kernel module information.
<code>--guest-search-path <path></code>	Specify the path of guest <code>vmlinux</code> and kernel sources copied on the local host. AMD uProf reads it to resolve the guest kernel module information.

Table 38. AMDuProfCLI Linux profile Command 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, the report for all categories is generated. Multiple instances of --category is allowed. Supported categories are:</p> <ul style="list-style-type: none"> • cpu: Generate a report specific to CPU Profiling. • mpi: Generate a report specific to MPI Tracing. • openmp: Generate a report specific to OpenMP Tracing. • trace: Generate a report specific to trace events. [os] deprecated • gputrace: Generate a report specific to GPU Tracing. • gpuprof: Generate a report specific to GPU Profiling. <p>Example:</p> <pre>--category cpu,mpi,trace,gputrace,gpuprof --category mpi --category cpu --category trace --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, function count will be generated for the entire profile duration.</p>
--branch-filter	<p>Use this option to capture LBR data. Specify the branch filter type:</p> <ul style="list-style-type: none"> • u: user branches • k: kernel branches • any: any branch type • any_call: any call branch • any_ret: any return branch • ind_call: indirect calls • ind_jump: indirect jumps • cond: conditional branches • call: direct calls <p>When the above filters are not set, the default filter type will be 'any'.</p> <p><i>Notes:</i></p> <ol style="list-style-type: none"> 1. When the above filters not set, the default filter type will be 'any'. 2. This option will work only with the PMC events. 3. This is applicable to per process and attach process profiling. However, it is not applicable to Java app profiling.

Table 38. AMDuProfCLI Linux profile Command Options

Option	Description
--vmlinux-path <path>	Path to the Linux kernel debug info file. If no path provided, it searches for the debug info file in the default download path.

6.9.4 Examples

Windows

- Launch application *AMDTClassicMatMul.exe* and collect the samples for CYCLES_NOT_IN_HALT and RETIRED_INST events and generate report:


```
C:\> AMDuProfCLI.exe profile -e cycles-not-in-halt -e retired-inst --interval 1000000
-o c:\Temp\cpuprof-custom AMDTClassicMatMul.exe
$ ./AMDuProfCLI.exe profile -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 IBS Samples and generate IMIX report:


```
AMDuProfCLI.exe profile --config ibs --imix -o c:\Temp\cpuprof-tbp AMDTClassicMatMul.exe
```
- Launch *AMDTClassicMatMul.exe* and perform Assess Performance profile for 10 seconds and generate report:


```
C:\> AMDuProfCLI.exe profile --config assess -o c:\Temp\cpuprof-assess -d 10
AMDTClassicMatMul.exe
```
- Launch *AMDTClassicMatMul.exe* and collect the IBS samples in the SWP mode and generate report sorted on ibs-op event:


```
C:\> AMDuProfCLI.exe profile --config ibs -a -s event=ibs-op -o c:\Temp\cpuprof-ibs-swp
AMDTClassicMatMul.exe
```
- Collect the TBP samples in SWP mode for 10 seconds and generate report:


```
C:\> AMDuProfCLI.exe profile -a -o c:\Temp\cpuprof-tbp-swp -d 10
```
- Launch *AMDTClassicMatMul.exe*, collect TBP with callstack sampling and generate report:


```
C:\> AMDuProfCLI.exe profile --config tbp -g -o c:\Temp\cpuprof-tbp AMDTClassicMatMul.exe
```
- Launch *AMDTClassicMatMul.exe*, collect TBP with callstack sampling (unwind FPO optimized stack) and generate report:


```
C:\> AMDuProfCLI.exe profile --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 and generate report sorted on pmcxc0 event:


```
C:\> AMDuProfCLI.exe profile -e event=pmcx76,interval=250000 -e
event=pmcxc0,user=1,os=0,interval=250000 -s event=pmcxc0 -o c:\Temp\cpuprof-tbp
AMDTClassicMatMul.exe
```

- Launch *AMDTClassicMatMul.exe* and collect the samples for IBS OP with an interval of 50000 and generate report sorted on ibs-op event:

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

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

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

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

```
C:\> AMDuProfCLI.exe profile --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 and generate report:

```
C:\> AMDuProfCLI.exe profile -e event=pmcx76,interval=250000 -e
event=pmcxc0,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 and generate report:

```
C:\> AMDuProfCLI.exe profile -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, Assess Performance samples, and generate report:

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

Linux

- Launch the application *AMDTClassicMatMul.bin*. Collect the samples for CYCLES_NOT_IN_HALT and RETIRED_INST events and generate report:

```
$ ./AMDuProfCLI profile -e cycles-not-in-halt -e retired-inst
--interval 1000000 -o /tmp/cpuprof-custom AMDTClassicMatMul-bin
$ ./AMDuProfCLI profile -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*. Collect the IBS samples and generate IMIX report from the raw data file:

```
$ ./AMDuProfCLI profile --config ibs --IMIX -o /tmp/cpuprof-tbp AMDTClassicMatMul-bin
```

- Launch *AMDTClassicMatMul-bin*. Perform Assess Performance profile for 10 seconds and generate report:

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

- Launch *AMDTClassicMatMul-bin*. Collect the IBS samples in the SWP mode and generate report sorted based on `ibs_op` event:

```
$ ./AMDuProfCLI profile --config ibs -a -s event=ibs_op -o /tmp/cpuprof-ibs-swp
AMDTClassicMatMul-bin
```

- Collect the TBP samples in SWP mode for 10 seconds and generate report:

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

- Launch *AMDTClassicMatMul-bin*. Collect TBP with callstack sampling and generate report:

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

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

```
$ ./AMDuProfCLI profile --config tbp --call-graph fpo:512 -o /tmp/uprof-tbp
AMDTClassicMatMulbin
```

- Launch *AMDTClassicMatMul-bin*. Collect the samples for PMCx076 and PMCx0C0 and generate report:

```
$ ./AMDuProfCLI profile -e event=pmcx76,interval=250000 -e
event=pmcxc0,user=1,os=0,interval=250000 -o /tmp/cpuprof-tbp AMDTClassicMatMul-bin
```

- Launch *AMDTClassicMatMul-bin*. Collect the samples for IBS OP with interval 50000 and generate report sorted on `ibs-op` event:

```
$ ./AMDuProfCLI profile -e event=ibs-op,interval=50000 -s event=ibs-op -o /tmp/cpuprof-tbp
AMDTClassicMatMulbin
```

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

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

- Collect OpenMP trace info of an OpenMP application, pass `-omp`, and generate report:

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

- Collect MPI profiling information and generate report:

```
$ mpirun -np 4 ./AMDuProfCLI profile --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 and generate report:

```
$ AMDuProfCLI profile -e event=pmcx76,interval=250000 -e
event=pmcxc0,interval=250000,callgraph -o /tmp/cpuprof-pmc AMDTClassicMatMul-bin
```

- Launch *AMDTClassicMatMul-bin*. Collect all the OS trace events and generate report:

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

- Launch *AMDTClassicMatMul-bin*. Collect the GPU Traces for Host Identity Protocol (HIP) domain and generate report:

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

- Launch *AMDTClassicMatMul-bin*. Collect the TBP samples, GPU Traces for the HIP domain, and generate report:

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

- Launch *AMDTClassicMatMul-bin*. Collect the GPU samples, OS Traces, and generate report:

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

6.10 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
```

6.10.1 Options

Following table lists the `info` command:

Table 39. AMDuProfCLI Info Command Options

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

Table 39. AMDuProfCLI Info Command Options

Option	Description
<code>--view-config <name></code>	Displays the details of the given view configuration used in the report generation option <code>report --view <name></code> . Use <code>info --list view-configs</code> command for the details on the supported data view configurations.
<code>--pmu-event <event></code>	Displays the details of the given pmu event. Use command <code>info --list pmu-events</code> for the list of supported pmc events.
<code>--system</code>	Displays the processor information of this system.

Following table lists the Linux specific info command options:

Table 40. AMDuProfCLI Info Command - Linux Specific Options

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

6.10.2 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
```

- Print the list of trace events:

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

Chapter 7 Performance Analysis

7.1 CPU Profiling

AMD uProf CPU profiler follows a statistical sampling-based approach to collect profile data to identify the performance bottlenecks in the application. A few high-level features to understand the CPU profiler capabilities are listed in this section:

- Profile data is collected using one of the following approaches:
 - Time 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
 - Launch App— launch an application and profile that process and its children.
 - System-wide — profile all the running processes and/or kernel.
 - Attach Process — Attach to an existing application (Native applications only)
- Profile mode
 - User/Kernel — profile data is collected when the application is running in User and/or Kernel mode.
- Supported Languages:
 - C, C++
 - Java
 - .NET (5.0, 6.0, and Framework)
 - FORTRAN
 - Assembly applications
- Supported Software Components
 - User-space applications
 - Dynamically linked/loaded modules
 - Drivers
 - OS kernel modules

- Profile data is attributed at various granularities:
 - Process, Thread, Load Module, Function, Source line, or Disassembly
 - C++ and Java in-lined functions

Note: uProf requires debug information from the compiler for correlating the profile data to functions and source lines.
- Data and Report Files:
 - Collected profile data initially stored to raw data files.
 - Processed profile data is stored to database files used for generating the CLI report or GUI visualization.
 - Profile report is saved to a comma-separated-value (CSV) format file that can be viewed using any spreadsheet viewer.
- AMDuProfCLI, the command-line-interface can be used to configure a profile run, collect the profile data, and generate the profile report.
 - Collect command to configure and collect the profile data.
 - Report command to process the profile data and to generate the profile report.
 - Profile command to collect the performance profile data, analyze it, and 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
 - Top-down and bottom-up call path — visualizations to explore the function call flow of an application for analyzing the time spent on functions and its callees.
 - Flame Graph — callstack visualizer as a flame graph
 - Call Graph — call stack and caller/callee visualizer in table format
 - HPC — to analyze OpenMP and MPI profile data
 - Timeline Visualizer — timeline views for MPI API trace and OS event trace information
 - Cache Analysis — to analyze the hot cache lines that are false shared

- Profile Control API
 - 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 Profiling** screen is displayed.

3. From the **Select Profiling** screen, select the **Predefined Configs** tab.

The following screen is displayed:

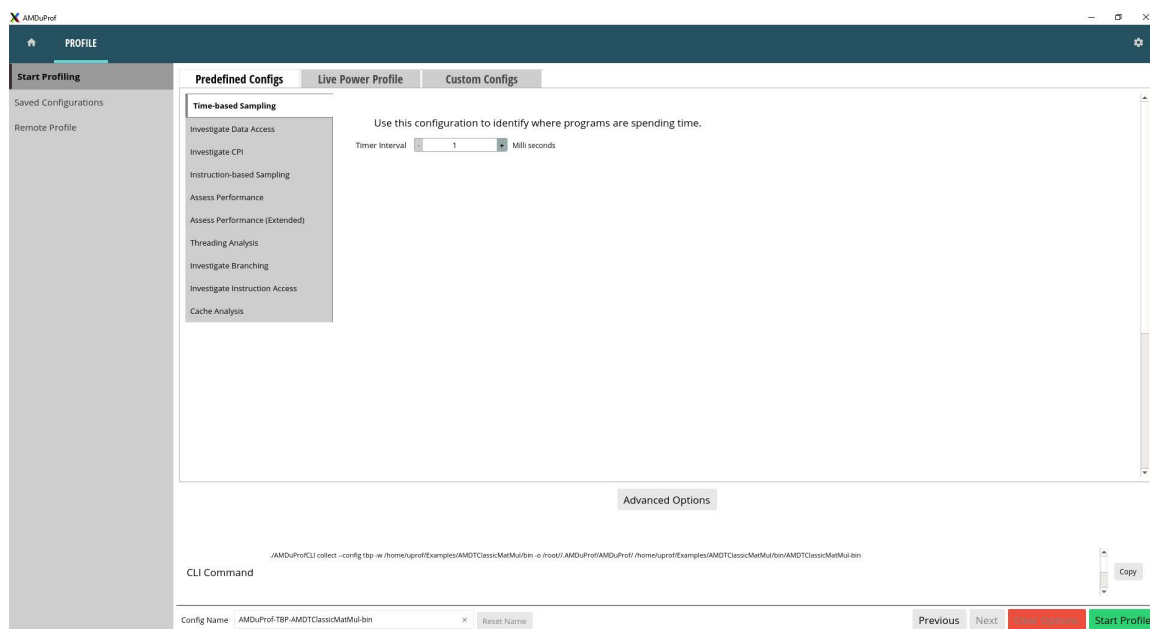


Figure 34. Time-based Profile – Configure

4. Select **Time-based Sampling** in the left vertical pane.

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” section for more information on this screen.
6. 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:

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. The hotspots are shown for the **Timer** samples. Refer the section “Overview of Performance Hotspots” 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” 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” 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” 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:

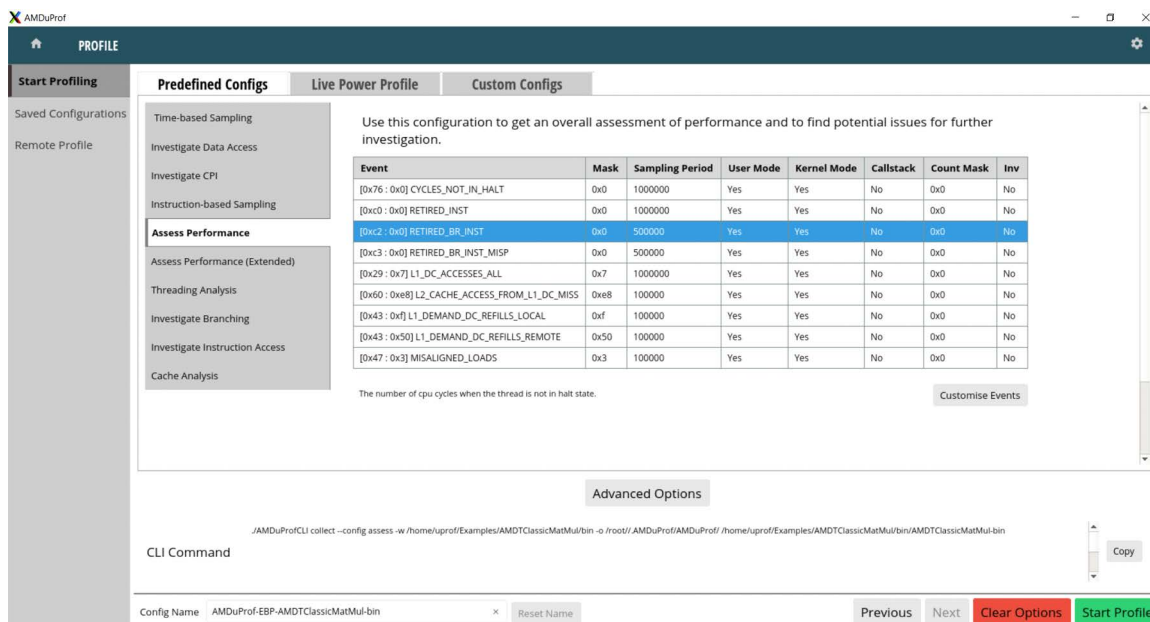


Figure 35. Event-based Profile – Configure

3. From the **Select Profiling** screen, select the **Predefined Configs** tab.
4. Select **Assess Performance** in the left vertical pane. Refer the section “Predefined Sampling Configuration” 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” 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” 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” 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 to the section “Process and Functions” for more information on this screen.
4. Double-click any entry on the **Functions** table in the **Grouped Metrics** screen to load the source tab for that function in **SOURCES** page. Refer the section “Source and Assembly” 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 Profiling** screen, select the **Predefined Configs** tab.
4. Select **Instruction-based Sampling** in the left vertical pane. Refer the section “Predefined Sampling Configuration” for IBS based predefined sampling configurations.

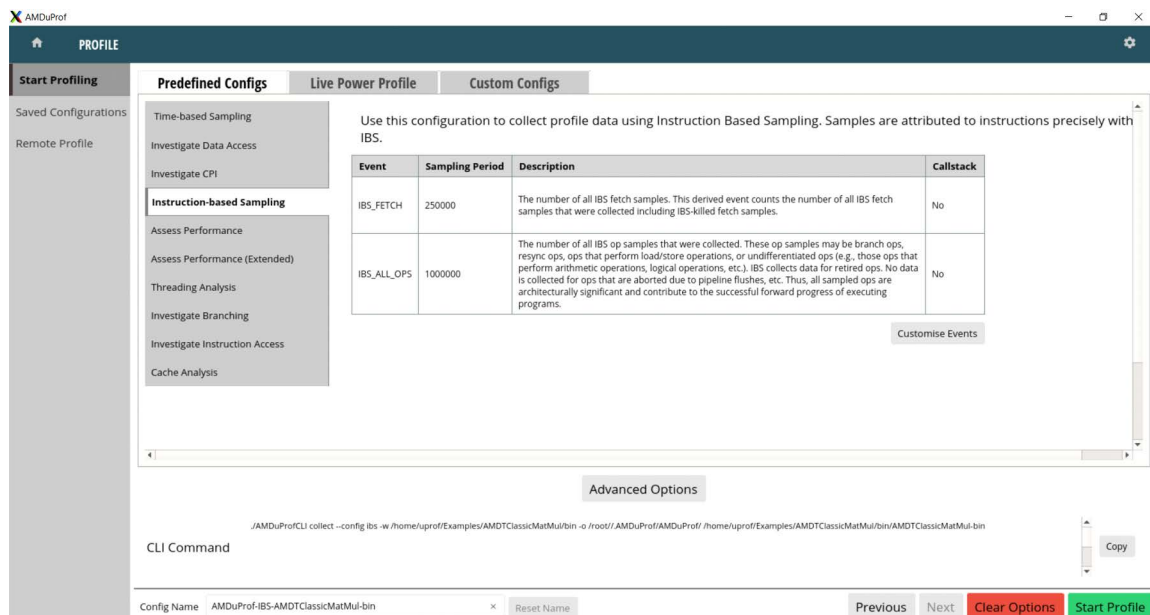


Figure 36. IBS Configuration

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

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

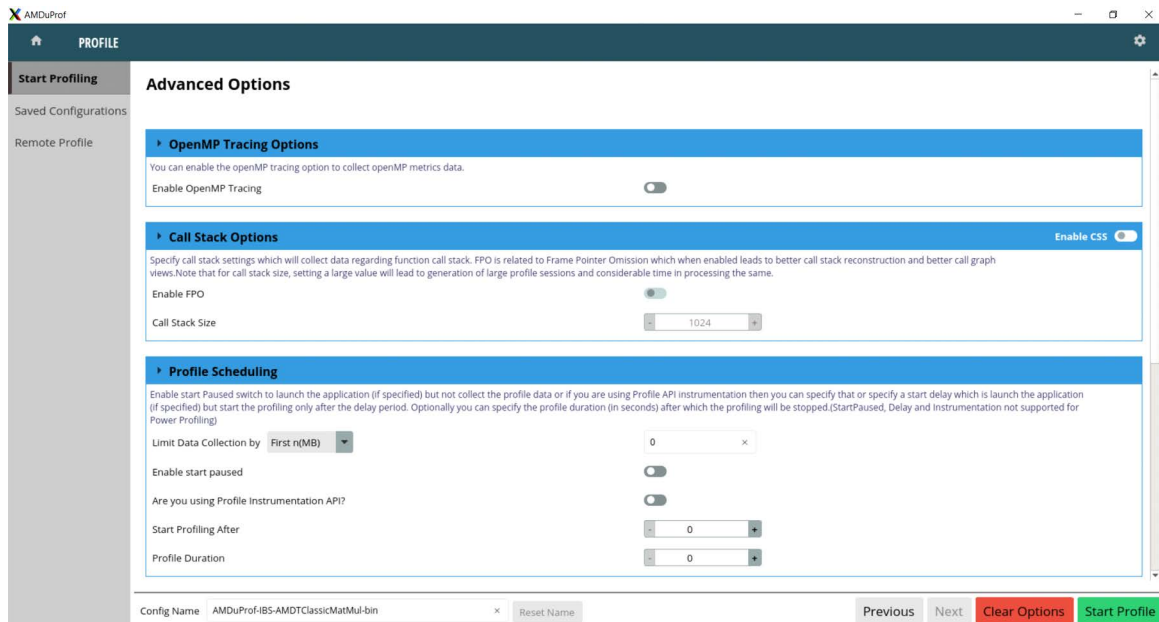


Figure 37. Start Profiling - Advanced Options

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

The following figure shows a sample flame graph:

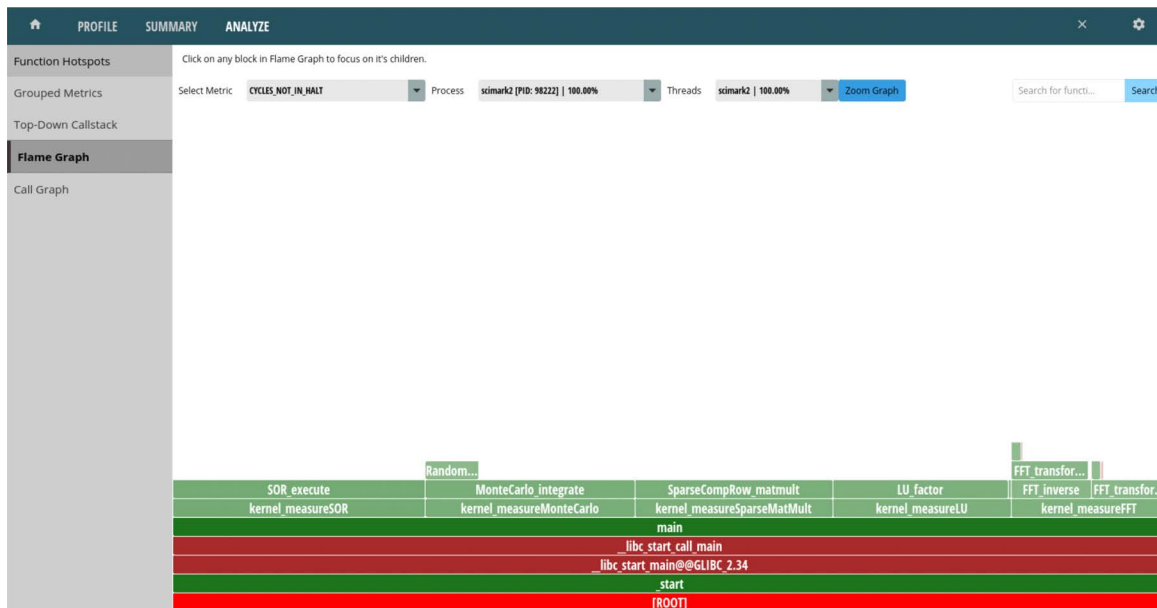


Figure 38. ANALYZE - Flame Graph

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 to the section “Call Graph” for more information on this screen.

The following figure shows a sample call graph:

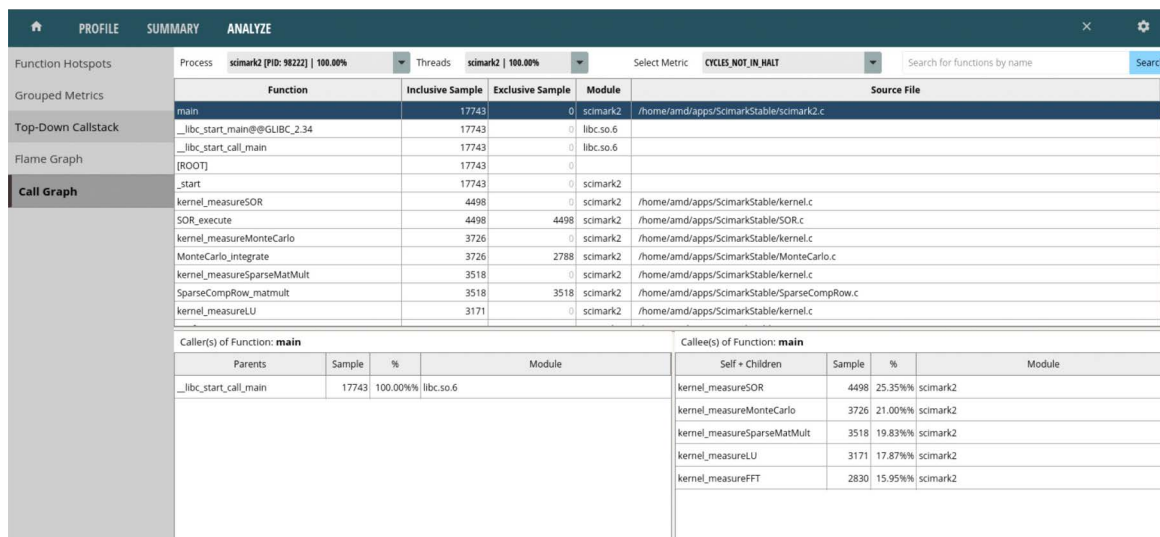


Figure 39. ANALYZE - Call Graph

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 to the section “Source and Assembly” for more information on source screen.

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

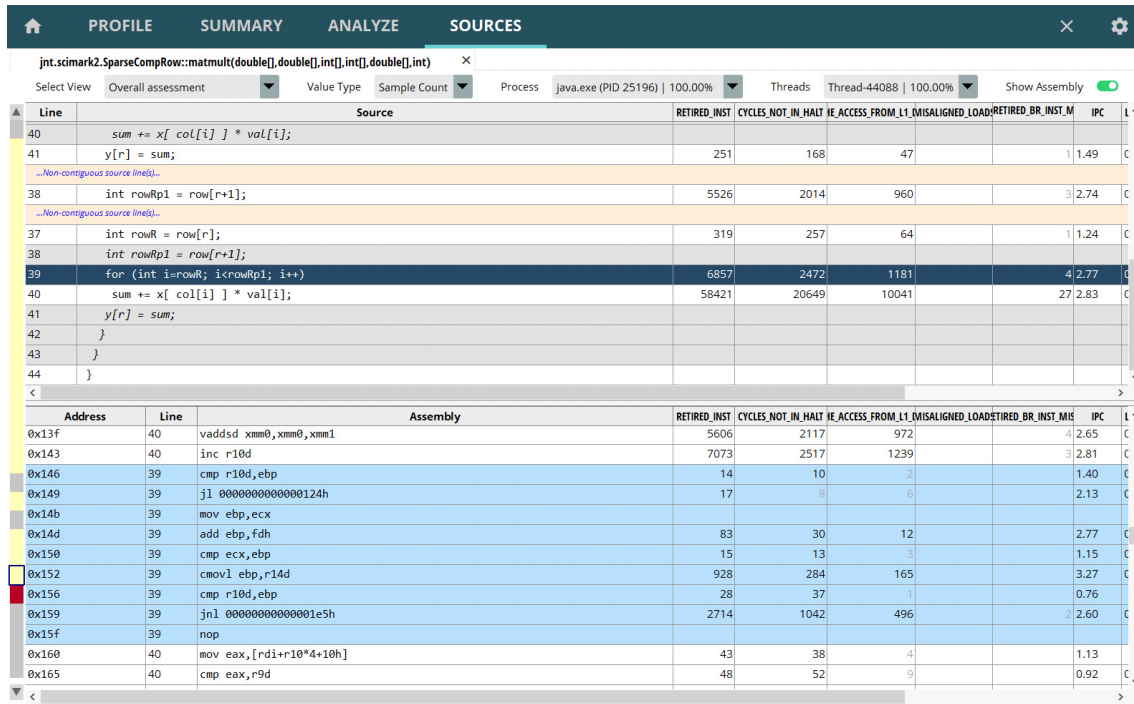


Figure 40. Java Method - Source View

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:

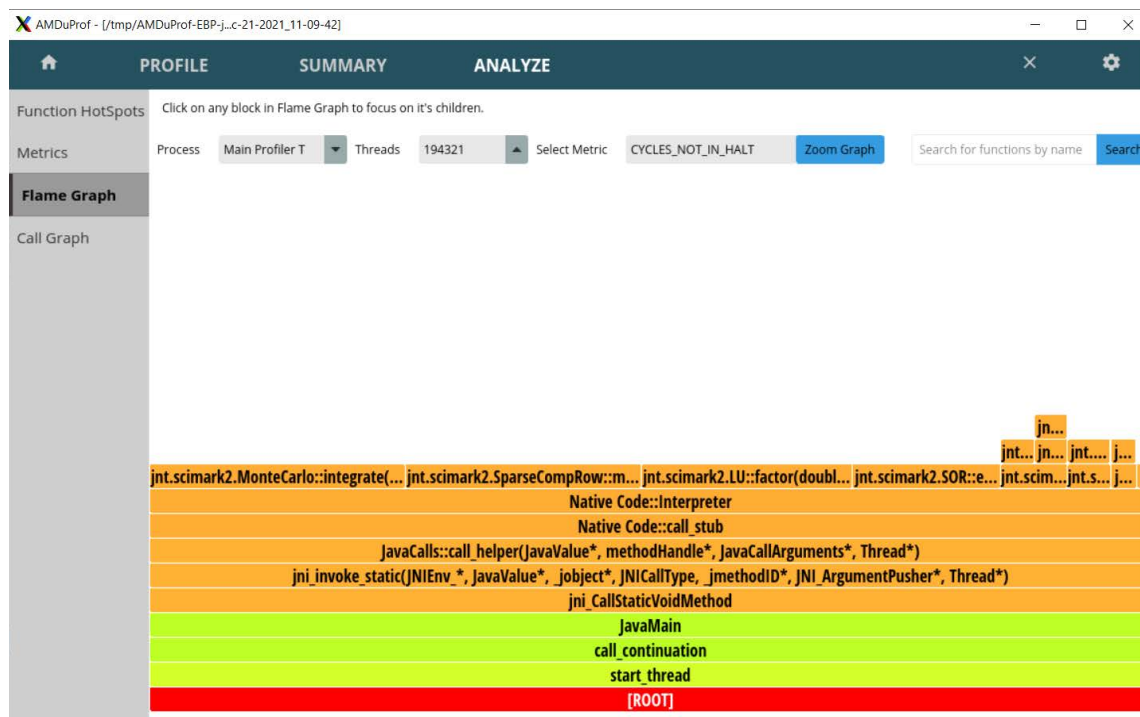


Figure 41. Java Application - Flame Graph

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 41. 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 **Predefined Configs** tab.
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 its 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
0x3d161f60c8	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	
0xa32878f0c8	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 42. 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_NB	IBS_NB	IBS_NB	IBS_NB	IBS_NB
0x3ecd4220c0	43639	6882	36757	38615	6540	36655	0	0	342	342	0	0
0xb617fa40c0	1902	182	1720	1393	170	1716	0	0	12	12	0	0
0xb617fa4000	7	7	0	0	7	0	0	0	0	0	0	0

Figure 43. 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_MI	IBS_LO	IBS_ST	IBS_NE	IBS_NE	IBS_NB	IBS_NB	IBS_NB	IBS_NB	IBS_NB	Function Name	Source File	Source Line
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 44. 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 42. Sort-by Metric

Sort-by Metric	Description
ldst-count	Total Loads and stores sampled
ld-count	Total Loads
st-count	Total Stores
cache-hitm	Loads that were serviced either from the local or remote cache (L3) and the cache hit state was Modified.

Table 42. Sort-by Metric

Sort-by Metric	Description
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 Configuration** screen is displayed.
3. From the **Select Profile Type** drop-down, select one of the following:
 - The **CPU Tracing Mode** drop-down consists of the options **OS Trace** and **User Mode Trace**. On Linux, **OS Trace** is enabled (with supported events) only in root/ADMIN mode and on

Windows, it's enabled with the supported event **Schedule**. **User Mode Trace** is enabled only for Application Analysis on Linux.

CPU Trace looks as follows:

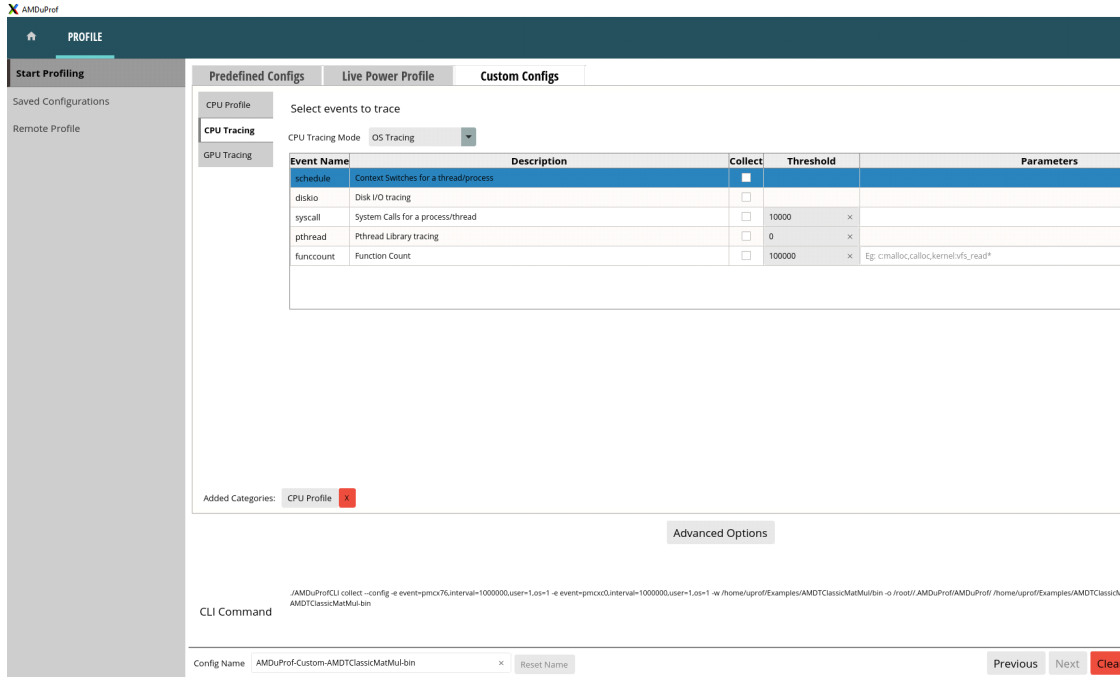


Figure 45. CPU Trace

- **GPU Trace** looks as follows:

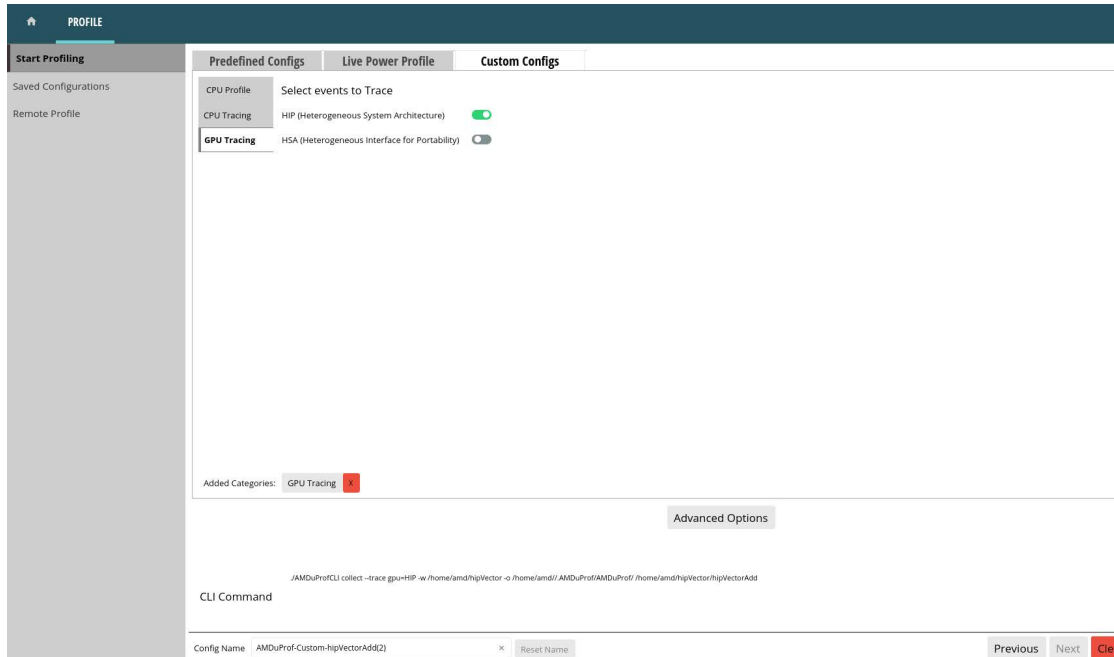


Figure 46. GPU Trace

Multiple categories from the custom configs can be added together, for example, **CPU Profile + CPU Trace**.

When multiple categories are selected, it will be mentioned below as breadcrumbs under **Added Categories** and you can deselect the unwanted categories. The corresponding CLI command will be generated below.

The custom configs screen will look similar to the following:

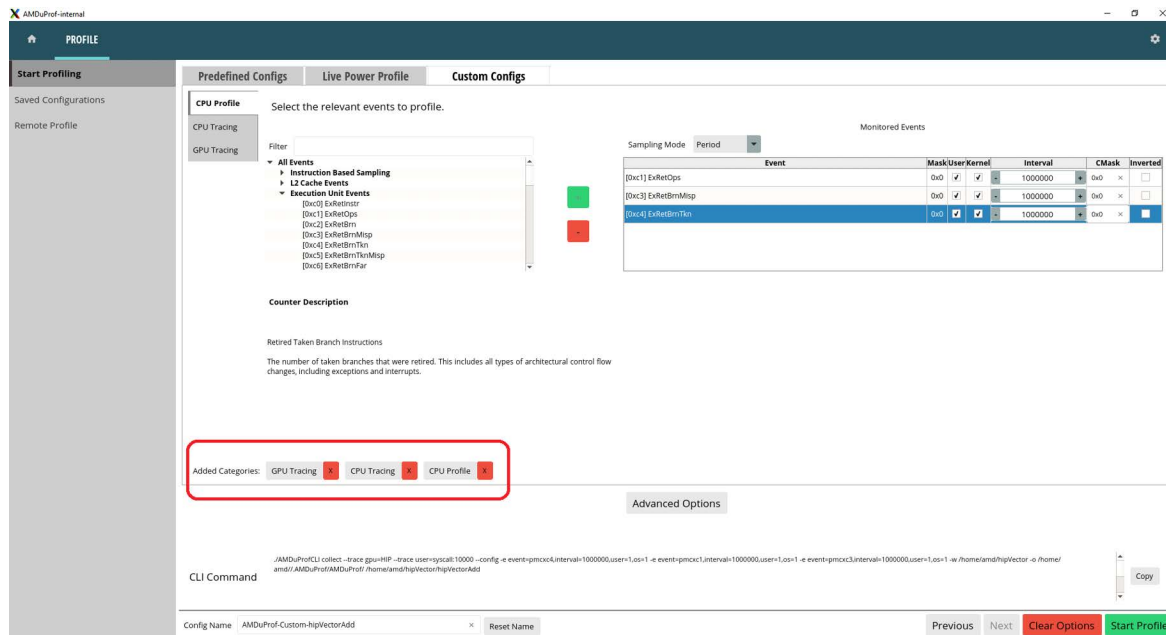


Figure 47. Custom Config - Added Categories

1. Select the **Custom Configs** tab and select **CPU Profile** from the left vertical pane.
2. 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” for more information on this screen.
3. 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:

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” 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” 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” 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” 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	CYCLES_NOT_IN_HALT	RETIRED_INST	IPC	CPI
java (PID 99124)	47981000000	212803000000	4.44	0.23
Thread-99128	47685000000	212229000000	4.45	0.22
C2 CompilerThre	241000000	518000000	2.15	0.47
C1 CompilerThre	34000000	43000000	1.26	0.79
Service Thread	7000000	8000000	1.14	0.88
VM Periodic Tas	7000000	0	0.00	0.00
C1 CompilerThre	5000000	3000000	0.60	1.67
java	1000000	2000000	2.00	0.50
G1 Young RemSet	1000000	0	0.00	0.00
C1 CompilerThre	0	0	0.00	0.00

Functions	Modules	CYCLES_NOT_IN_HALT	RETIRED_INST	IPC	CPI
jnt.scimark2.MonteCarlo::integrate(int)	jnt.scimark2.coi	12930000000	74554000000	5.77	0.17
jnt.scimark2.SOR::execute(double,double[][],int)	jnt.scimark2.coi	9709000000	7574000000	0.78	1.28
jnt.scimark2.SparseCompRow::matmult(double[],double[],int)	jnt.scimark2.coi	9147000000	56985000000	6.23	0.16
jnt.scimark2.LU::factor(double[][],int[])	jnt.scimark2.coi	7776000000	23581000000	3.03	0.33
jnt.scimark2.FFT::transform_internal(double[],int)	jnt.scimark2.coi	7381000000	46863000000	6.35	0.16
jnt.scimark2.kernel::CopyMatrx(double[][],double[][],int)	jnt.scimark2.coi	184000000	638000000	3.47	0.29
Native Code::Interpreter	jnt.scimark2.coi	141000000	185000000	1.31	0.76
Native Code::libmSin	jnt.scimark2.coi	128000000	896000000	7.00	0.14
jnt.scimark2.FFT::inverse(double[])	jnt.scimark2.coi	53000000	91000000	1.72	0.58
jnt.scimark2.LU::factor(double[][],int[])	jnt.scimark2.coi	35000000	165000000	4.71	0.21
jnt.scimark2.SOR::execute(double,double[][],int)	jnt.scimark2.coi	22000000	33000000	1.50	0.67
jnt.scimark2.SparseCompRow::matmult(double[],double[],int)	jnt.scimark2.coi	20000000	113000000	5.65	0.18
jnt.scimark2.Random::nextDouble()	jnt.scimark2.coi	19000000	85000000	4.47	0.22

Figure 48. CPI Metric - Threshold-based Performance

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 Branch Analysis

AMD “Zen4” processors support Last Branch Record (LBR) CPU feature that is useful for branch analysis. Use uProf CLI to collect and generate the branch analysis report.

Branch analysis is supported only on Linux platform.

Notes:

1. *PMC event must be enabled for LBR sample collection. If no PMC event is passed, PMCX0C0 event is enabled during LBR sample collection.*
2. *Branch analysis is not supported for Java apps.*

Example

Collect the LBR info:

```
$ AMDuProfCLI collect --branch-filter -o /tmp/ ./ScimarkStable/scimark2_64static
```

Generate branch analysis report:

```
$ AMDuProfCLI report --detail -i /tmp/AMDuProf-scimark2_64static-Custom_May-15-2023_21-05-56
```

Sample Report

The report generated contains a section for branch analysis. A sample screenshot for branch analysis summary is as follows:

TAKEN BRANCH ANALYSIS SUMMARY											
OVERHEAD(%)	SAMPLES	MISPREDICT(%)	MISPREDICT COUNT	SOURCE FUNCTION	TARGET FUNCTION	SOURCE LINE	TARGET LINE	SOURCE MODULE	TARGET MODULE	PROCESS	
32.45	2498527	0.02	434	LU_factor	LU_factor	/home/deesin/home/deesin/scimark2_64static	scimark2_64stat	scimark2_64stat	scimark2_64stat	(PID:468487)	
18.02	1387483	0.01	77	SOR_execute	SOR_execute	/home/deesin/home/deesin/scimark2_64static	scimark2_64stat	scimark2_64stat	scimark2_64stat	(PID:468487)	
14.46	1113561	0	21	SparseCompRow_matmult	SparseCompRow_matmult	/home/deesin/home/deesin/scimark2_64static	scimark2_64stat	scimark2_64stat	scimark2_64stat	(PID:468487)	
5.35	411872	0.01	52	FFT_transform_internal	FFT_transform_internal	/home/deesin/home/deesin/scimark2_64static	scimark2_64stat	scimark2_64stat	scimark2_64stat	(PID:468487)	
4.4	338909	0	0	Random_nextDouble	Random_nextDouble	/home/deesin/home/deesin/scimark2_64static	scimark2_64stat	scimark2_64stat	scimark2_64stat	(PID:468487)	
3.54	272318	0	0	SparseCompRow_matmult	SparseCompRow_matmult	/home/deesin/home/deesin/scimark2_64static	scimark2_64stat	scimark2_64stat	scimark2_64stat	(PID:468487)	
2.45	188579	0.01	25	Random_nextDouble	MonteCarlo_integrate	/home/deesin/home/deesin/scimark2_64static	scimark2_64stat	scimark2_64stat	scimark2_64stat	(PID:468487)	
2.38	183204	0	8	FFT_transform_internal	FFT_transform_internal	/home/deesin/home/deesin/scimark2_64static	scimark2_64stat	scimark2_64stat	scimark2_64stat	(PID:468487)	
2.34	180103	0.01	25	Random_nextDouble	MonteCarlo_integrate	/home/deesin/home/deesin/scimark2_64static	scimark2_64stat	scimark2_64stat	scimark2_64stat	(PID:468487)	
2.33	179614	0	0	MonteCarlo_integrate	Random_nextDouble	/home/deesin/home/deesin/scimark2_64static	scimark2_64stat	scimark2_64stat	scimark2_64stat	(PID:468487)	
2.02	155651	0	0	MonteCarlo_integrate	Random_nextDouble	/home/deesin/home/deesin/scimark2_64static	scimark2_64stat	scimark2_64stat	scimark2_64stat	(PID:468487)	

Figure 49. Branch Analysis Summary

The branch analysis summary table comprises of the following columns:

- **OVERHEAD (%)**: Indicates which branching was mostly taken. Calculated as: $(\text{SAMPLES} * 100) / (\text{Total SAMPLES})$.
- **SAMPLES**: Shows the number of samples collected for the branch. This does not indicate the actual branches taken.
- **MISPREDICT (%)**: Indicates ratio of mispredicts occurred for the branch. Calculated as: $((\text{MISPREDICT COUNT}) * 100 / \text{SAMPLES})$
- **MISPREDICT COUNT**: Shows the number of branch mis-predicted samples collected for the branch.
- **SOURCE FUNCTION**: Shows the function from where the branch was taken.
- **TARGET FUNCTION**: Shows the function into which the branch was taken.
- **SOURCE LINE**: Shows the file path and line number (from where the branch was taken) of the SOURCE FUNCTION.
- **TARGET LINE**: Shows the file path and line number (into which the branch was taken) of the TARGET FUNCTION.
- **SOURCE MODULE**: Shows the module name of the SOURCE FUNCTION.
- **TARGET MODULE**: Shows the module name of the TARGET FUNCTION.
- **PROCESS**: Shows the name and PID of the process.

7.12 Export Session

The CLI option `--export-session` helps to generate a compressed archive containing essential session files. The compressed archive can be easily transported to other system and the GUI can be used for analyzing the performance data.

This feature streamlines the process of transferring and utilizing session files across multiple systems, enhancing accessibility and enabling smooth workflow continuity.

Steps

Complete the following steps to export a session:

1. Generate compressed archive with `translate`, `report`, or `profile` command.
A `.zip` file is generated.
2. Copy the `.zip` file to another system and decompress it.

The decompressed session directory can be imported to GUI for data visualization and analysis. To import the decompressed session and to analyze the performance data, refer to the section [“Importing Profile Database”](#).

Common Usage

- Generate compressed archive with 'translate' command:

```
/AMDuProfCLI translate <options> --export-session <options> -i <session_dir>
```

- Generate compressed archive with 'report' command:

```
./AMDuProfCLI report <options> --export-session <options> -i <session_dir>
```

- Generate compressed archive with 'profile' command:

```
./AMDuProfCLI profile <options> --export-session <options>
```

Example

Launch the application `AMDTCClassicMatMul.exe` and collect the Time-Based Profile (TBP) samples and generate a report with the export session option enabled:

```
AMDuProfCLI.exe profile --config tbp --export-session -o c:\Temp\cpuprof-tbp  
AMDTCClassicMatMul.exe
```

7.13 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).

IMIX has the following limitations:

- The IMIX view or report is supported only for IBS profile type.
- If any module/binary has less than 10 samples, it is not shown in the IMIX report. Extremely less number of samples are not useful for IMIX analysis.
- Linux kernel module .ko files are not shown in the IMIX view or report.

Chapter 8 Performance Analysis (Linux)

This chapter explains the Linux specific performance analysis models.

8.1 Threading Analysis

You can use threading analysis to identify how efficiently an application uses:

- Processor cores
- Contention among the threads due to synchronization
- CPU utilization of threads
- Runtime and wait time analysis of application threads

Limitations

- It is not supported when an application is statically linked with libc and libpthread.
- The behavior is undefined when an application uses the clone system call for thread or process creation instead of pthread_create() or fork().
- It is not supported with system-wide profiling and attach process.
- Supported only on AMD “Zen3” and AMD “Zen4” platforms. On other platforms, use the custom configuration to collect the data.

8.1.1 Threading Analysis Using CLI

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

Collect Threading Data

CLI command to collect the threading data:

```
$ AMDuProfCLI collect --config threading -o /tmp/threading-analysis/ /home/app/classic_lock  
...  
Generated data files path: /tmp/threading-analysis/AMDuProf-classic_lock-Threading_Jun-13-  
2023_06-00-23
```

This command will launch the program to collect the profile and trace data. When the launched application is executed, 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/threading-analysis/AMDuProf-classic_lock-Threading_Jun-13-2023_06-00-23
```

Collect Threading and System Calls

Enable the system call collection along with threading to get the IO, syscall, and block time of each thread. CLI command to collect threading and system calls tracing:

```
$ AMDuProfCLI collect --config threading --trace user=syscall -o /tmp/threading-analysis/ /home/app/classic_lock
...
Generated data files path: /tmp/threading-analysis/AMDuProf-classic_lock-Threading_Jun-13-2023_06-00-23
```

Collect Threading and Context Switch

Enable the context switch collection (root access is required) for accurate wait time analysis:

```
$ sudo AMDuProfCLI collect --config threading --trace os=schedule -o /tmp/threading-analysis/ /home/app/classic_lock
...
Generated data files path: /tmp/threading-analysis/AMDuProf-classic_lock-Threading_Jun-13-2023_06-00-23
```

Collect with Custom Config

Example command to collect the CPU cycles event in frequency mode (with frequency set as 100Hz), pthread synchronizing APIs trace data and system calls:

```
$ sudo AMDuProfCLI collect -e event=pmcx76,umask=0,frequency=100 --trace user=pthread,syscall -o /tmp/threading-analysis/ /home/app/classic_lock
...
Generated data files path: /tmp/threading-analysis/AMDuProf-classic_lock-Threading_Jun-13-2023_06-00-23
```

Generate Profile Report

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

```
$ sudo AMDuProfCLI report -i /tmp/threading-analysis/AMDuProf-classic_lock-Threading_Jun-13-2023_06-00-23
...
Generated report file: /tmp/threading-analysis/AMDuProf-classic_lock-Threading_Jun-13-2023_06-00-23/report.csv
```

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

APPLICATION PERFORMANCE SNAPSHOT					
Number Of Threads	48				
Elapsed Time	14.0079 seconds				
Serial Execution Time	0.053427 seconds				
Parallel Execution Time	11.3603 seconds				
Total Time	545.413 seconds				
Run Time	179.76 seconds				
Wait Time	364.969 seconds				
Sleep Time	0.683631 seconds				
IO Time	20.2477 seconds				
Block Time	0.00639536 seconds				
CPU TRACING REPORT					
Note: Time represents the total runtime & wait time of all the threads					
SYSTEM CALL SUMMARY					
System Call	Count	Total Time(seconds)	Min Time(seconds)	Max Time(seconds)	Avg Time(seconds)
__libc_poll	14341	182.618	1.00E-05	11.1591	0.0127339
epoll_wait	2028	181.541	1.05E-05	10.8013	0.0895172
__GI_readv	1097288	12.9824	1.00E-05	0.000757105	1.18E-05
openat	180455	4.41378	1.00E-05	0.00127476	2.45E-05
writv	123107	2.48682	1.00E-05	0.00052576	2.02E-05
THREAD SUMMARY					
Process	Thread	Time(seconds)	Wait Time(seconds)	Wait Time(% From Thread Elapsed Time)	
simpleFoam(42680)	simpleFoam(42740)	11.455	11.454	99.99	
simpleFoam(42680)	simpleFoam(42741)	11.4494	11.4462	99.97	
simpleFoam(42694)	simpleFoam(42779)	11.397	11.396	99.99	
simpleFoam(42686)	simpleFoam(42780)	11.3965	11.3956	99.99	
simpleFoam(42684)	simpleFoam(42783)	11.3909	11.3901	99.99	
WAIT OBJECT SUMMARY					
Wait Object	Thread	Wait Count	Total Wait Time(sec)	Wait Time (% From C File)	
CV@0x7ffd770aa548	"simpleFoam(42694)"	1	0.0426698	100	
CV@0x7ffeb4918bc8	"simpleFoam(42707)"	1	0.0426543	100	
CV@0x7ffd682cf658	"simpleFoam(42718)"	1	0.0425546	100	
CV@0x7ffd6d10c518	"simpleFoam(42717)"	1	0.0424495	100	
CV@0x7ffeca1817a8	"simpleFoam(42684)"	1	0.0423476	100	

Figure 50. Trace Report

From the report, the application performance snapshot provides the following details:

- Number of threads/Thread count: Total number of threads created by the application.
- Elapsed time: Total elapsed time of the application.
- Serial time: Total time of the application when only one thread is running.
- Parallel time: Total time of the application when two or more threads are running.
- Run time: Total run time of all threads. If context switch records are collected, the total run time will be total time of all the threads executing in CPU. Otherwise, total run time = total time - (total wait time + total sleep time)

- **Wait time:** Total wait time of all the threads. Wait time is calculated as follows:
 - **Threading config** (`--config threading`): Total time spent by a thread in pthread synchronization APIs and wait system calls. Refer section 8.1.2 and 8.1.3 for traced synchronization APIs and wait system calls.
 - **Custom config** (`--trace user=syscall`): Total time spent by a thread in wait system calls.
 - **Custom config** (`--trace user=pthread`): Total time spent by a thread in pthread synchronization APIs.
 - **Custom config** (`--trace os/--trace os=schedule`): Total time of all the threads when a thread is not in CPU. It uses the context switch records to identify whether thread is in CPU or not.
- **Sleep time:** Total time spent by all the threads in sleep system calls. Refer to section 8.1.3 for sleep system calls that are traced.
- **IO time:** Total time spent by all the threads in IO system calls. Refer to section 8.1.3 for IO system calls that are traced.
- **Block time:** Total time spent by all the threads in blocking the system calls. When application makes this type of system call, there is no guarantee that the application will be blocked. So, this block time will be added to the total run time too. Refer to section 8.1.3 for block system calls that are traced.

Summary Report Sections

- **System call summary:** Provides the system call count, total time spent by the application on a system call. Helps identify the system calls consuming most of the time and that can be optimized if the system calls blocking in nature.
- **Thread summary:** Provides the total run time, wait time of each thread, and wait time percentage with respect to the total time of thread. Helps identify if a thread is using the core effectively or not. Wait time of threads should be low for an optimized application.
- **Wait object summary:** pthread synchronization object wait count and total wait time due to this synchronization object. Helps identify the object responsible for most of the wait time.
- Import the profiled session in GUI and navigate to **Analyze > Thread Timeline** for better visualization, thread timeline analysis, pthread synchronization object analysis, and call stack analysis.

8.1.2 pthread Synchronization APIs

List of thread synchronization APIs traced when pthread trace event is enabled:

- pthread_mutex_lock
- pthread_mutex_trylock
- pthread_mutex_timedlock
- pthread_cond_wait
- pthread_cond_timedwait
- pthread_cond_signal
- pthread_cond_broadcast
- pthread_rwlock_rdlock
- pthread_rwlock_tryrdlock
- pthread_rwlock_timedrdlock
- pthread_rwlock_wrlock
- pthread_rwlock_trywrlock
- pthread_rwlock_timedwrlock
- pthread_spin_lock
- pthread_spin_trylock
- pthread_barrier_wait
- sem_wait
- sem_trywait
- sem_timedwait
- pthread_create
- pthread_join
- pthread_cancel
- pthread_yield
- pthread_exit

8.1.3 libc System Call Wrapper APIs

List of libc functions traced when syscall event is enabled:

Sleep APIs

- sleep
- pause
- sigtimedwait
- nanosleep
- sigsuspend
- clock_nanosleep
- sigwait
- usleep
- sigwaitinfo

Wait APIs

- poll
- ppoll
- select
- pselect
- epoll_wait
- epoll_pwait
- wait
- waitpid
- waitid
- wait3
- wait4

IO APIs

- create
- open
- openat
- read
- pread
- readv
- preadv
- preadv2
- write
- pwrite
- writev
- pwritev
- pwritev2
- lseek
- sendfile
- copy_file_range
- truncate
- ftruncate
- readahead
- close

Blocking APIs

- flock
- fsync
- sync
- syncfs
- fdatsync
- sync_file_range
- accept
- accept4
- recv
- recvfrom
- recvmsg
- recvmmsg
- send
- sendto
- sendmsg
- sendmmsg
- mq_send
- mq_timedsend
- mq_receive
- mq_timedreceive
- msgsnd
- msgrcv
- semget
- semop
- semtimedop
- semctl
- splice
- vmsplice
- msync
- fcntl
- ioctl
- epoll_create
- epoll_create1
- epoll_ctl

Other APIs

- socket
- bind
- listen
- connect
- socketpair
- mq_notify
- mq_getattr
- mq_setattr
- mq_close
- mq_unlink
- msgget
- msgctl
- pipe
- pipe2
- shmat
- shmctl
- shmget
- shmdt
- fork
- vfork
- alarm
- system
- kill
- killpg
- brk
- sbrk
- mlock
- munlock
- mlock2
- mlockall
- munlockall
- mmap
- munmap
- move_pages
- mprotect
- mremap
- process_vm_readv
- process_vm_writev
- acct
- chroot
- dup
- dup2
- dup3
- fallocate
- ioperm
- iopl
- mount
- prctl
- ptrace
- sigaction
- swapon
- swapoff
- tee
- umount
- umount2
- unshare
- vhangup

8.1.4 Timeline Analysis GUI in Linux

To configure threading analysis from the GUI:

1. Navigate to the **Select Profile Configuration** screen.
2. Select **Predefined Configs** from the tab.
3. Select **Threading Analysis** from the left vertical pane.

Profile data collected from CLI or GUI can be visualized in GUI by importing the session. On importing, the following section (Thread Timeline) is displayed on the *ANALYZE* page.

Time-series data is plotted in timelines per entity (thread, rank, device, and so on). Trace data (if collected) will only be plotted when you zoom into the timeline to address data size related scalability issues (trace data can have millions of records which will not be visually legible if plotted together). The entire view is broadly separated in three vertical parts, top data selectors, middle timelines, and bottom filters. You can use the timeline as follows:

- Hover the cursor over a timeline to view a vertical line containing the tool-tip for a specific entity, showing relevant details, and the current timestamp.
- If callstack data is collected, click at any point in the timeline to bring up the callstack of the corresponding entity in the bottom pane.

***Note:** There can be multiple callstacks at a given timestamp as sampling data is coarse-grained.*

- If CPU profile data is collected, click and drag the mouse over the timeline to select a region across all timelines and brings up the **Function Hotspot** within the selected time range.

- Zoom-in/out horizontally into the timelines using one of the following:
 - The mouse wheel.
 - Pressing CTRL and +/- keys on the keyboard to Zoom-in/out respectively.
 When the timeline is zoomed in, trace data (if present) is displayed.

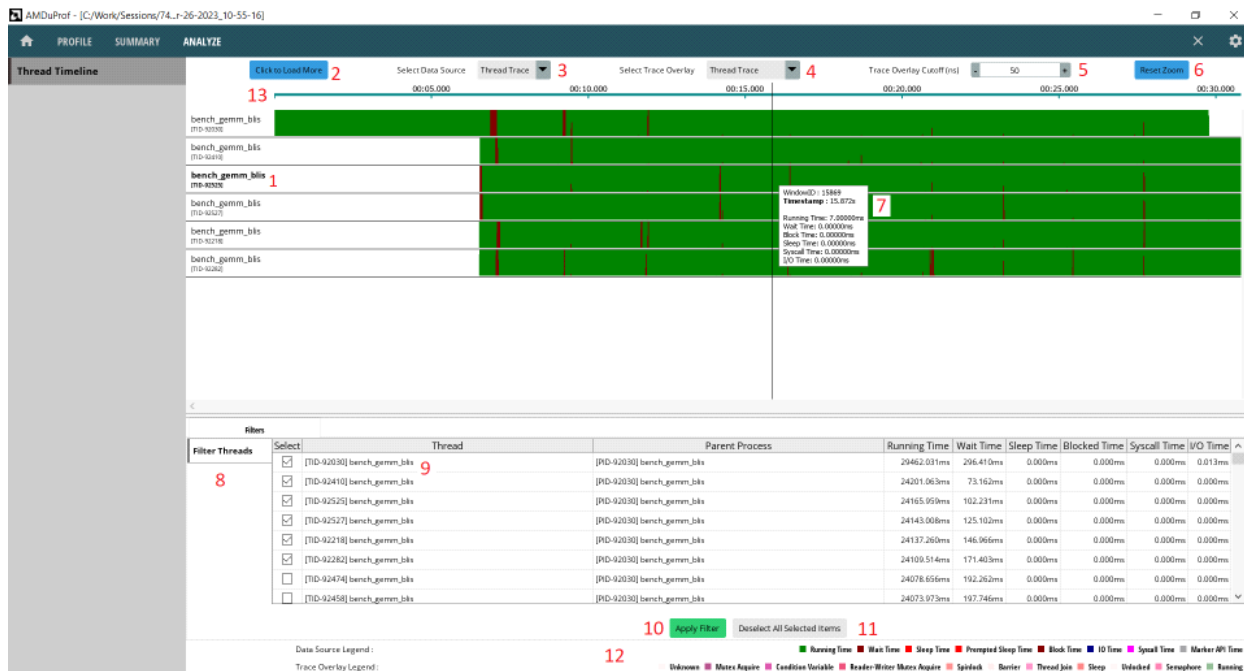


Figure 51. Timeline Analysis GUI in Linux

The timeline section consists of:

1. Name of each thread in timeline with **Thread ID**.
2. **Click to Load More** button which loads more threads. By default, only a small number of thread timelines are loaded to limit the resource consumption. This button enables loading the next set of thread timelines. The next set is determined by the entries in the table below the timeline.

3. Select the **Data Source** drop-down to enable selection of data to display on the timeline. Different types of data source are as follows:

- **CPU Utilization:** Plots the timeline for the CPU utilization (in %) per thread at a per second interval. To collect sufficient such data points, the total profile duration should be greater than or equal to 10 seconds. This is enabled only for the **Threading Analysis** configuration.
- **Memory Consumption:** Plots the timeline for the memory consumption (in MB) categorized as physical and virtual memory consumed. This is enabled only for the **Threading Analysis** configuration.
- **Context Switches:** Plots the timeline for both voluntary context switches count (sleep, yield, and so on) or involuntary context switches count (OS scheduler triggered context switch). This is enabled only for the **Threading Analysis** configuration.
- **CPU Profile Samples:** Plots the timeline for the CPU sample collected for the CPU events. The following events are supported:

Table 43. Supported CPU Events

Events	Availability
Retired Instructions	PMC event RETIRED_INSTRUCTIONS is collected.
Cycles not in Halt	PMC event CYCLES_NOT_IN_HALT is collected.
Op Cycles	IBS op event is collected with 'count cycles' unit mask.
CPU Time	Time-based profiling is performed.

- **Thread Trace:** Plots the timeline based on OS trace data which can either originate from eBPF Tracing or User-mode Tracing. The trace data is categorized and aggregated at certain intervals to generate time-series plotted in timelines. The following categories are created:

Table 44. CPU Trace Categories

Category	Description
Wait Time	Total time spent in synchronization objects, that is, mutex, condition variable, semaphore, locks, barriers, latches, and so on
Sleep Time	Total time spent in sleep syscalls.
Running Time	If only user-mode tracing is enabled: Running Time = Total Time – (Wait Time + Sleep Time). If eBPF tracing is enabled, then Running Time is total active time in CPU: Running Time = Total Time – Sleep Time (from context switch records)
Block Time	Total Time spent in blocking syscalls, that is, select, epoll, poll, wait, accept, and so on.
I/O Time	Total Time spent in I/O syscalls, that is, read, write, pread, pwrite, and so on.
Syscall Time	Total time spent on all traced syscalls – (Block Time + I/O Time)

4. The **Select Trace Overlay** drop-down enables selection of the type of trace data to display.
 - **Don't Show Trace**: Trace data will not be loaded in the timeline.
 - **Thread State**: Shows the current state of thread from eePBF or User-mode tracing. In the former, thread state is inferred from BPF data. In the later, thread state is treated as **Running** if Running Time > 0, otherwise, **Sleeping**.
 - **Thread Trace**: Displays traces for the traced libpthread functions, such as pthread_mutex_lock, pthread_mutex_trylock, and so on.
 - **Syscalls**: Displays traces for traced syscall in the specific region of the timeline.
5. **Trace Cutoff** can be used to specify a duration in nanoseconds, which acts as a cutoff to load the trace data, that is, any traced function which takes less than the specified nanoseconds will not be displayed.
6. Click the **Reset Zoom** button to reset any zoom performed earlier.
7. Hover over any timeline to view the tool-tip containing the relevant data along with timestamp. If trace data is also present, the relevant traced functions with start time and duration.
8. **Filter Threads/Ranks** enables you to filter which thread's (or rank's) timelines must be displayed. By default, the timelines are sorted internally and the first 6 are loaded. However, from the table, you can select the required threads and clicking **Apply Filter** to apply the changes. If CPU profile data is collected, highlighting functions or modules is also possible. Each function is assigned a random color, which can be modified and highlighted in the timeline (implies there are samples from the function/module).
9. Each entry in the filter table has the necessary data, that is, name, parent object, and samples/trace times aggregated across the profile.
10. Click the **Apply Filter** button to apply a custom selection of entities or highlight entities in timeline.
11. Click **Deselect selected Items** to deselect all the entries in the filtering table except the first one. This is useful when a custom selection is required but all timelines are already loaded.
12. At the bottom of the filtering pane, timeline legend is displayed, which helps in identifying how each type of 'data source' or 'trace' is mapped to which color.
13. The **Show Core Transition** button is disabled by default and works only when the CPU profiling data is collected. When enabled, a red line is displayed in each timeline to signify when a thread changes the core.
14. If any configuration is profiled with CSS enabled, select **Threading Analysis > Select Data Source > CPU Profile Samples**. The callstack section will be enabled only if you select a valid samples region.

Note: Time-series data (from **Select Data Source**) will be plotted as a line graph, where the x-axis is time and y-axis the height implies how close to the maximum value it reached. For trace records, the height is always total height of the timeline. However, the width varies based on the duration of the traced function.

8.2 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 45. 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.

8.2.1 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:

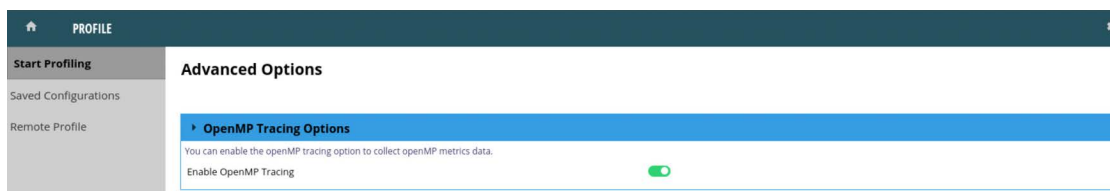


Figure 52. Enable OpenMP Tracing

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:

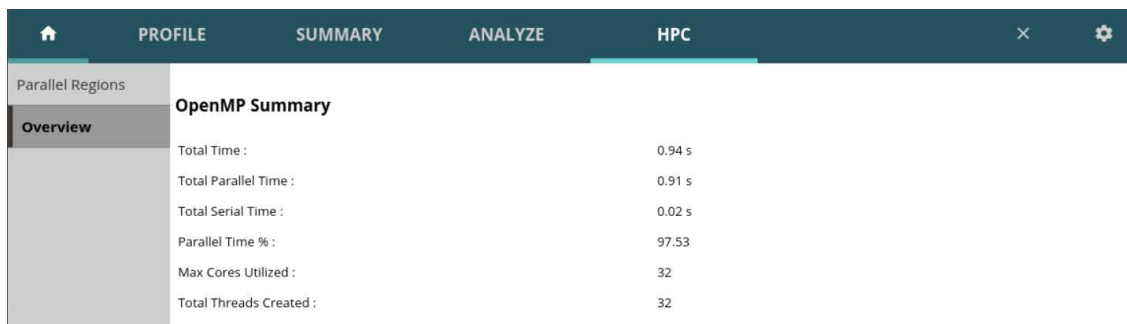


Figure 53. HPC - Overview

- **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	Avg Overhead T	Avg Work Time	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
LinkedCells:get	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:upd	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 54. HPC - Parallel Regions

8.2.2 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.

Modes of tracing OpenMP events are:

- **Full Tracing:** All the OpenMP events are traced in full tracing. Use the following command to perform full OpenMP tracing:

```
./AMDuProfCLI collect --trace openmp=full -o /tmp/myapp_perf <openmp-app>
```

- **Basic Tracing:** Only the events which are required for the high level report generation are traced. The size of trace data collected is less as compared to the full tracing mode. This is the default mode. Use the following command to perform basic OpenMP tracing:

```
./AMDuProfCLI collect --trace openmp=basic -o /tmp/myapp_perf <openmp-app>
```

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	0.446554	1	Static	0.469849
collatz_sequence_compute\$omp\$parallel_for:4@collatz-sequence-omp-10pr.c:34	0.000239	0.050082	4	0.000239	0	0.021354	0.456124	1	Static	0.477718
OpenMP THREAD METRIC (collatz_sequence_compute\$omp\$parallel_for:4@collatz-sequence-omp-10pr.c:34)										
ThreadNum	Threadid	Idle Time	Sync Time	Overhead	Work Time					
0	27842	0	0	0.064491	0.411899					
1	27845	0.00001	0	0.026767	0.449614					
2	27846	0.000008	0	0.012695	0.463688					
3	27847	0.000009	0	0.000005	0.476377					
OpenMP THREAD METRIC (collatz_sequence_compute\$omp\$parallel_for:4@collatz-sequence-omp-10pr.c:34)										
ThreadNum	Threadid	Idle Time	Sync Time	Overhead	Work Time					
0	27842	0	0	0.060944	0.410298					
1	27845	0.000007	0	0.023169	0.448067					
2	27846	0.000006	0	0.009212	0.462025					
3	27847	0.000006	0	0.000005	0.471232					
OpenMP THREAD METRIC (collatz_sequence_compute\$omp\$parallel_for:4@collatz-sequence-omp-10pr.c:34)										
ThreadNum	Threadid	Idle Time	Sync Time	Overhead	Work Time					
0	27842	0	0	0.060453	0.409315					

Figure 55. 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.

OpenMP trace data can be collected in Linux and the session can be imported to GUI or CLI on Windows.

8.2.3 Environment Variables

AMDUPROF_MAX_PR_INSTANCES – Set the max number of parallel regions to be traced. The default value is 2000.

8.2.4 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.

8.3 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 46. 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 8.6 and 9
	CentOS 8

8.3.1 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
```

8.3.2 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 reported 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
```

8.3.3 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 167.
2. To import the profile database, refer “Importing Profile Database” on page 72.

8.3.4 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.

8.4 Profiling Support on Linux for perf_event_paranoid Values

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

Table 47. Profiling perf_event_paranoid Values on Linux

Config	Profile Scope	perf_event_paranoid Values			
		-1	0	1	2
Time Based Profiling	Specific Application or Process	Y	Y	Y	Y
Time Based Profiling	Kernel, Hypervisor	Y	Y	Y	N
Time 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

8.5 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.

8.6 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

8.6.1 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
```

8.6.2 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-dbg-sym-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)-dbg-sym
```

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
```

8.6.3 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.

8.6.4 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.

8.6.5 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
```

8.6.6 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.

8.7 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 48. 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 7. To validate the BCC Installation, run the script `sudo AMDuProfVerifyBpflInstallation.sh`.

8.7.1 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 Reac	Total Write C	Total Read S	Total Write Si	Avg IO Latenc	Read Bandwid	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 Si	Total Write Si	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 56. Disk I/O Summary Tables

Analyze Trace Data with GUI

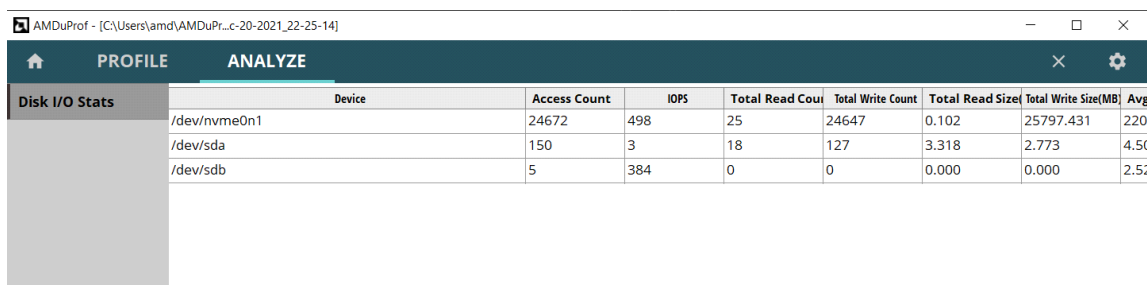
To visualize the trace data collected using CLI, the collected raw profile and trace data should be processed using CLI `translate` command and then it can be imported in the GUI.

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:



PROFILE		ANALYZE						
Disk I/O Stats	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	4.56
	/dev/sdb	5	384	0	0	0.000	0.000	2.52

Figure 57. ANALYZE - Block I/O Stats

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

8.8 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 49. 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 5.5 to be installed. For the steps to install AMD ROCm, refer section “Installing ROCm” on page 6.

Note: Tracing might not work as expected on '5.2.1 or older' 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-5.5.0/
```

- 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-5.5.0/lib
```

8.8.1 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/
```

The behavior is undefined when the GPU profile collection is interrupted or the launch application is killed from other terminal.

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-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 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)
JacobIterationKernel(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 58. GPU Tracing Report

For more information on GPU tracing from GUI, refer to the section 7.8.1.

8.9 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 5.5 to be installed. For the steps to install AMD ROCm, refer section “Installing ROCm” on page 6

Note: Profiling might not work as expected on '5.2.1 or older' 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 50. 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 50. 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 51. 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

8.9.1 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/
```

The behavior is undefined when the GPU profile collection is interrupted or the launch application is killed from other terminal.

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	Avg Elapsed Ti	Elapsed Ti	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<fl	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_COUN	GRBM_GUI_AC	SQ_WAVES	TCC_HIT_5	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<fl	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 A	Max LDS A	Min LDS A	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<fl	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 59. GPU Profile Report

8.10 Other OS Tracing Events

Apart from the OS events that are listed in section “Kernel Block I/O Analysis” on page 172, following OS events can also be traced along with CPU sampling-based profiles:

Table 52. 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>
funccount	Trace the functions provided with the option <code>--func</code> .

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 7. To validate the BCC Installation, run the script `sudo AMDuProfVerifyBpfInstallation.sh`.

8.10.1 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-0sTrace_Dec-09-2021_12-19-27
...
Generated report file: /tmp/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 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 Allocati	Memory Deallocation Count	
/home/amd/SamplePrograms/Scima	0.097412	2.37E-05	7	6	

Figure 60. Pagefault and Memory Allocation Summary

8.10.2 Tracing Function Call Count using CLI

`funcccount` 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 27 on page 88.

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 Avg Time(seconds)
main	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable	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 Avg Time(seconds)
kernel_measureFFT	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable	ScimarkStable(118908)	1	8.69741	8.69741	8.69741
RandomVector	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable	ScimarkStable(118908)	1	0.0020856	0.0020856	0.0020856
FFT_transform	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable	ScimarkStable(118908)	1	0.000123624	0.000123624	0.000123624
FFT_transform_internal	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable	ScimarkStable(118908)	1	0.000117512	0.000117512	0.000117512
new_Random_seed	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable	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 Avg Time(seconds)
FFT_transform_internal	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable	ScimarkStable(118908)	41	0.00456172	0.000100861	0.000171625
FFT_transform	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable	ScimarkStable(118908)	24	0.00273375	0.000101231	0.000173209
FFT_inverse	/home/amd/ScimarkStable/Linux_x64_Debug/ScimarkStable	ScimarkStable(118908)	18	0.00208564	0.000108716	0.00012703

Figure 61. 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
```

For more information on GPU tracing from GUI, refer to the section 7.8.1.

8.11 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.

Fortran bindings are configured and built while compiling the MPI implementations. You can enable/disable the Fortran bindings based on your need for Fortran language support.

Refer the following options to disable/enable the Fortran bindings:

- OpenMPI

```
--enable-mpi-fortran[=VALUE]
--disable-mpi-fortran
```

By default, OpenMPI will attempt to build all the 3 Fortran bindings: mpif.h, mpi module, and mpi_f08 module.

- MPICH

```
--disable-fortran
```

By default, the Fortran bindings are enabled. You can use this option to disable it.

Support Matrix

Table 53. Support Matrix

Component	Supported Versions
MPI Spec	MPI v3.1
MPI Libraries	Open MPI v4.1.4, MPICH v4.0.3, ParaStation MPI v5.6.0, and Intel® MPI 2021.1
OS	<ul style="list-style-type: none"> • Ubuntu: 18.04 LTS, 20.04 LTS, and 22.04.04 LTS • RHEL: 8.6 and 9 • CentOS 8.4
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 88.

8.11.1 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 54. 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

Table 54. List of Supported MPI APIs for Light-weight Tracing

MPI_Isend	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:

MPI FUNCTIONS SUMMARY							
Function	Min Time(seconds)	Max Time(seconds)	Average Time(seconds)	MPI Time(%)	Volume(Byte)	Calls	Total Time(seconds)
MPI_Probe	0.00001	0.15412	0.0236	0.56106	0	14	0.33037
MPI_Iprobe	0	0.15545	0.00005	3.48577	0	39557	2.05253
MPI_Wait	0.0004	0.28788	0.09301	8.6874	0	55	5.11541
MPI_Barrier	0	0.13712	0.05841	6.3484	0	64	3.73814
MPI_Recv	0	0.04478	0.00607	0.14428	899680	14	0.08495
MPI_Irecv	0	0.00001	0	0.00036	25433040	72	0.00021
MPI_Send	0.00002	0.21753	0.04939	1.1742	899680	14	0.6914
MPI_Isend	0	0.0001	0.00001	0.00109	25433040	72	0.00064
MPI_Reduce	0.00001	0.20347	0.0487	1.9848	1152	24	1.16871
MPI_Allreduce	0.00001	1.98697	0.21382	61.00609	5312	168	35.92229
MPI_Bcast	0.00002	0.1231	0.0555	6.03232	60849496	64	3.55202

MPI RANK SUMMARY							
Rank	PID	MPI Time(seconds)	MPI Time(%)	Wait Time(seconds)	Call Count	Volume(Bytes)	
0	1646816	4.93464	8.3804	0.6901	6719	2E+07	
1	1646829	9.85067	16.72919	0.56098	16343	1E+07	
2	1646828	6.03492	10.24898	0.49534	1191	1E+07	
3	1646806	8.42714	14.31164	0.48829	23500	1E+07	
4	1646819	5.10993	8.67809	0.39546	912	1E+07	
5	1646830	9.1953	15.61618	1.13217	12698	1E+07	
6	1646818	6.2113	10.54852	0.60311	13873	1E+07	
7	1646817	9.11923	15.48701	0.74996	5622	1E+07	

Figure 62. LWT Report

8.11.2 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 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.
 - **Total Time (seconds):** Total time spent in the API in all the ranks.
- 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.
 - **Total Time (seconds):** Total time spent in the API in all the ranks.

The list of supported MPI APIs is as follows:

Table 55. 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

Table 55. MPI APIs

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
MPI_Sendrecv_replace	MPI_Neighbor_Allgather	MPI_Ineighbor_Allgatherv	MPI_Dist_graph_create_adjacent
MPI_Recv_Init	MPI_Neighbor_Allgatherv	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:4.0.2	
MPI Std Version		4
Thread Model	MPI_THREAD_SINGLE	
MPI COMMUNICATOR SUMMARY (All Ranks)		
Ranks	Communicator Size	Elapsed Time(seconds)
0;1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;	16	0.003844
0;3;6;9;12;15;	6	0.002388
0;1;3;4;6;7;9;10;12;13;15;	11	0
1;4;7;10;13;	5	0.002872
1;2;4;5;7;8;10;11;13;14;	10	0
2;5;8;11;14;	5	0.002861

Figure 63. 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 64. 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	Total Time(seconds)
MPI_Cart_create	0.000661161	0.577336	0.192807	5.29	0	16	3.08491
MPI_Wait	5.09E-05	0.21472	0.0810626	8.63	0	62	5.02588
MPI_Send	1.24E-05	0.274551	0.0757055	1.82	899680	14	1.05988
MPI_Probe	1.31E-05	0.154153	0.0141356	0.34	0	14	0.197899
MPI_Recv	3.50E-06	0.0485917	0.00687159	0.17	899680	14	0.0962022
MPI_Test	6.29E-06	0.222162	8.55E-05	4.48	0	30528	2.60932
MPI_Iprobe	1.00E-07	0.105778	5.98E-05	3	0	29276	1.74939
MPI_Isend	1.47E-06	0.000238472	1.48E-05	0	25433040	72	0.0010647
MPI_Irecv	6.92E-07	7.88E-06	2.59E-06	0	25433040	72	0.0001866

Figure 65. MPI 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 66. MPI Communication Matrix

COLLECTIVE EVENTS SUMMARY								
Function	Min Time(seconds)	Max Time(seconds)	Average Time(seconds)	MPI Time(%)	Input Volume(Bytes)	Output Volume(Bytes)	Calls	Total Time(seconds)
MPI_Allreduce	8.86E-06	1.98372	0.215983	62.27	2656	2656	168	36.2852
MPI_Barrier	4.35E-06	0.110859	0.0603471	6.63	0	0	64	3.86221
MPI_Bcast	7.65E-06	0.207049	0.055552	6.1	53243309	7606187	64	3.55554
MPI_Reduce	1.13E-05	0.177126	0.0309416	1.27	144	1008	24	0.742599

Figure 67. MPI Collective API Summary Table

8.11.3 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 185. Import the report to GUI as shown in the following figure to analyze the trace data:

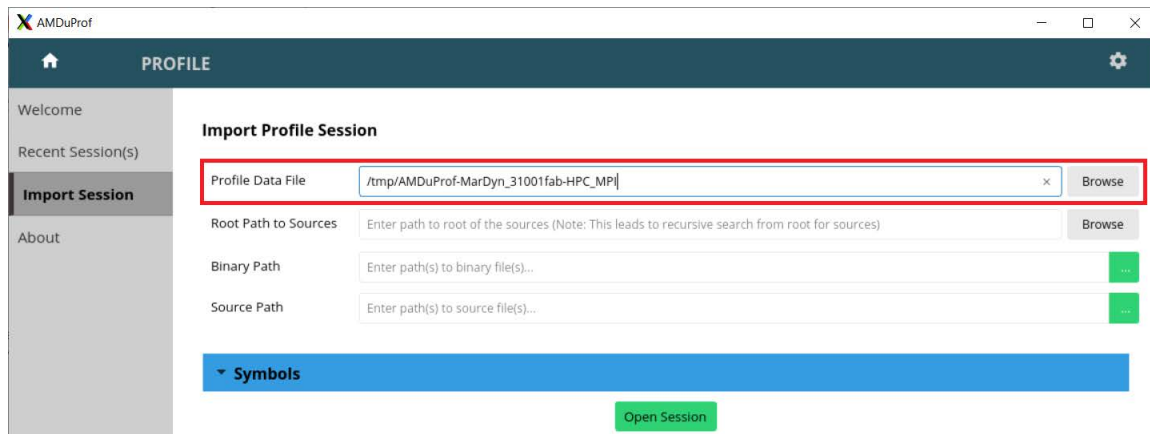


Figure 68. Import Profile Session

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 Communication Matrix** 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:

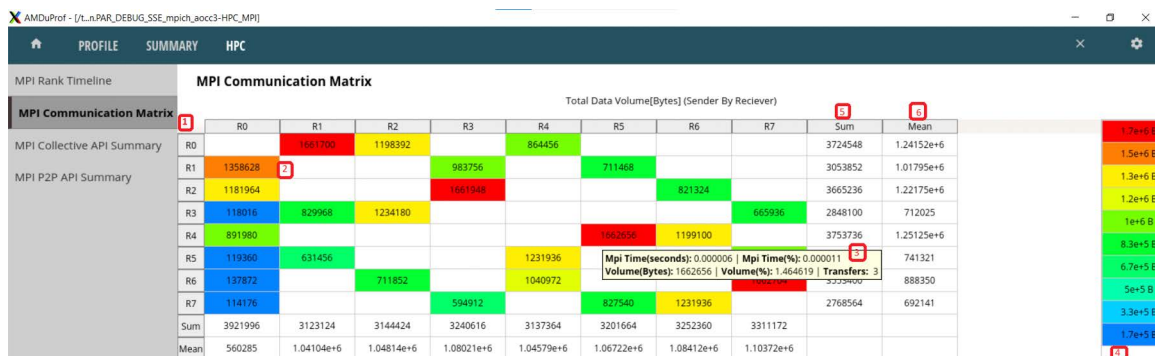


Figure 69. MPI Communication Matrix

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 Timeline

Navigate to **HPC > MPI Rank Timeline** to view to MPI Ranks timeline. This view shows the MPI activities in the timeline graph as follows:

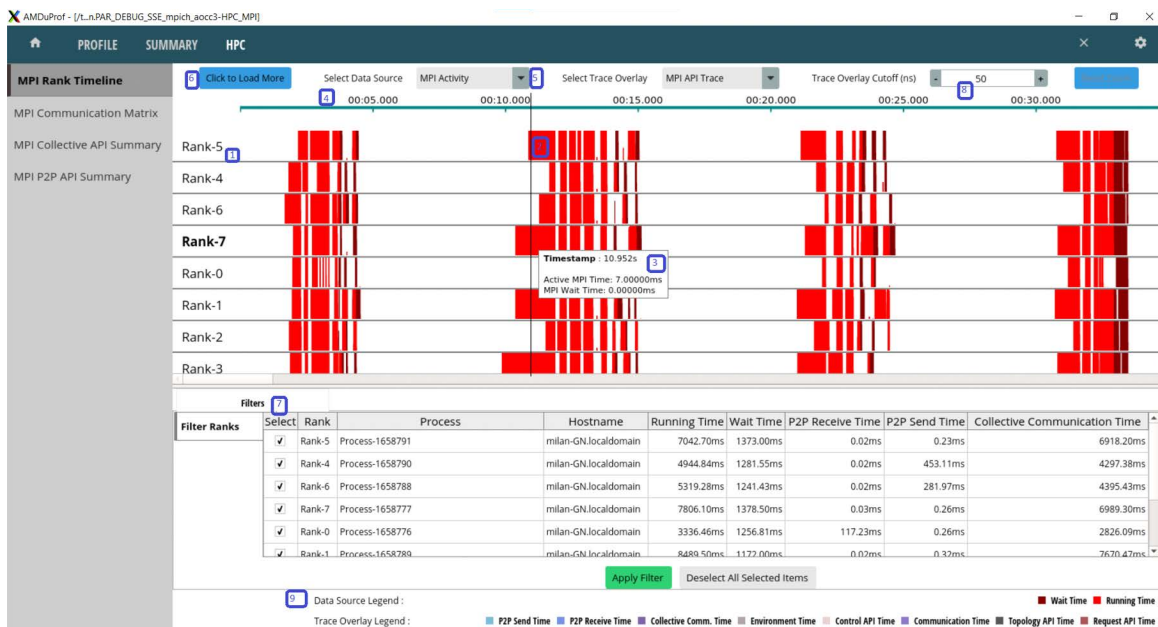


Figure 70. MPI Rank Timeline

In the above screenshot:

1. Rank ID
2. ~~To display~~ Graph of one of the following depending on the selected data source:
 - MPI API Activity (running or waiting)
 - MPI data transfer activity (receiving or sending)
 - MPI APIs called
3. Tool-tip shows more information about the MPI activity.
4. Displays the time range.
5. To select the data source MPI Activity. For more information, refer to the section “MPI Data Source”.
6. To load more rank details.
7. To filter the ranks from the view.

8. **Trace Overlay Cutoff** can be used to specify a duration in nanoseconds, which acts as a cutoff to load the trace data, that is, any traced data source which takes less than the specified nanoseconds will not be displayed.
9. Color coding legends for data source and trace overlay.

Analyzing MPI P2P API Summary

Navigate to **HPC > MPI P2P API Summary**. This view summarizes the P2P APIs called by the application as follows:

MPI Rank Timeline	Function	Min Time	Max Time	Average Time	MPI Time(%)	Volume(Bytes)	Calls
MPI Communication Matrix	MPI_Cart_create	0.00s	0.58s	0.19s	5.29	0	16
	MPI_Wait	0.00s	0.21s	0.08s	8.63	0	62
	MPI_Send	0.00s	0.27s	0.08s	1.82	899680	14
MPI Collective API Summary	MPI_Probe	0.00s	0.15s	0.01s	0.34	0	14
MPI P2P API Summary	MPI_Recv	0.00s	0.05s	0.01s	0.17	899680	14
	MPI_Test	0.00s	0.22s	0.00s	4.48	0	30528
	MPI_Jprobe	0.00s	0.11s	0.00s	3.00	0	29276
	MPI_Send	0.00s	0.00s	0.00s	0.00	25433040	72
	MPI_Recv	0.00s	0.00s	0.00s	0.00	25433040	72

Figure 71. MPI P2P API Summary

Analyzing MPI Collective API Summary

Navigate to **HPC > MPI Collective API Summary**. This view summarizes the collective APIs called by the application as follows:

MPI Rank Timeline	Function	Min Time	Max Time	Average Time	MPI Time(%)	Input Volume(Bytes)	Output Volume(Bytes)	Calls
MPI Communication Matrix	MPI_Allreduce	0.00s	1.98s	0.22s	62.27	2656	2656	168
	MPI_Barrier	0.00s	0.11s	0.06s	6.63	0	0	64
	MPI_Bcast	0.00s	0.21s	0.06s	6.10	53243309	7606187	64
MPI Collective API Summary	MPI_Reduce	0.00s	0.18s	0.03s	1.27	144	1008	24

Figure 72. MPI Collective API Summary

MPI Data Source

Supported list of MPI data source is as follows:

- An MPI Activity that classifies MPI APIs into either "waiting" APIs (MPI_Barrier, MPI_Wait, MPI_Waitall, MPI_Waitany, or MPI_Waitsome) or "active" APIs (all the other MPI functions).

- MPI APIs can be classified as follows:

P2P Send	P2P Receive	Collective Communication
MPI_BSEND	MPI_IMRECV	MPI_ALLGATHER
MPI_BSEND_INIT	MPI_IRECV	MPI_ALLGATHERV
MPI_IBSEND	MPI_MRECV	MPI_ALLREDUCE
MPI_IRSEND	MPI_RECV	MPI_ALLTOALL
MPI_ISEND	MPI_RECV_INIT	MPI_ALLTOALLV
MPI_ISSEND		MPI_ALLTOALLW
MPI_RSEND		MPI_BARRIER
MPI_RSEND_INIT		MPI_BCAST
MPI_SEND		MPI_GATHER
MPI_SEND_INIT		MPI_GATHERV
MPI_SENDRECV		MPI_IALLGATHER
MPI_SENDRECV_REPLACE		MPI_IALLGATHERV
MPI_SSEND		MPI_IALLREDUCE
MPI_SSEND_INIT		MPI_IALLTOALL
		MPI_IALLTOALLV
		MPI_IALLTOALLW
		MPI_IBARRIER
		MPI_IBCAST
		MPI_IGATHER
		MPI_IGATHERV
		MPI_IREDUCE
		MPI_IREDUCE_SCATTER
		MPI_ISCAN
		MPI_ISCATTER
		MPI_ISCATTERV
		MPI_REDUCE
		MPI_REDUCE_SCATTER
		MPI_SCAN
		MPI_SCATTER
		MPI_SCATTERV

Control API	Request API	Communication API
MPI_PCONTROL	MPI_CANCEL MPI_START MPI_STARTALL MPI_TEST MPI_TESTALL MPI_TESTANY MPI_TESTSOME MPI_WAIT MPI_WAITALL MPI_WAITANY MPI_WAITSOME MPI_IMPROBE MPI_IPROBE MPI_MPROBE MPI_PROBE	MPI_COMM_CREATE MPI_COMM_DUP MPI_COMM_DUP_WITH_INFO MPI_COMM_SPLIT MPI_COMM_SPLIT_TYPE MPI_COMM_SET_NAME MPI_INTERCOMM_CREATE MPI_INTERCOMM_MERGE MPI_CART_CREATE MPI_CART_SUB MPI_GRAPH_CREATE MPI_DIST_GRAPH_CREATE MPI_DIST_GRAPH_CREATE_ADJACENT

Topology API	Environment API
MPI_NEIGHBOR_ALLGATHER MPI_NEIGHBOR_ALLGATHERV MPI_NEIGHBOR_ALLTOALL MPI_NEIGHBOR_ALLTOALLV MPI_NEIGHBOR_ALLTOALLW MPI_INEIGHBOR_ALLGATHER MPI_INEIGHBOR_ALLTOALL MPI_INEIGHBOR_ALLGATHERV MPI_INEIGHBOR_ALLTOALLV MPI_INEIGHBOR_ALLTOALLW	MPI_ABORT MPI_FINALIZE MPI_INIT MPI_INIT_THREAD

- MPI Data Transfer which classifies MPI P2P Send/Receive and plots the volume of data transferred at the given time interval.

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
 - Package level metrics – RAPL based Package Power and Temperature
- 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 56. 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 56. 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 57. 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 58. 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 59. 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 59. 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 60. 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 Configuration** screen, select the **Live Power Profile** tab.

All the live profiling options and available counters are displayed in the respective panes as follows:

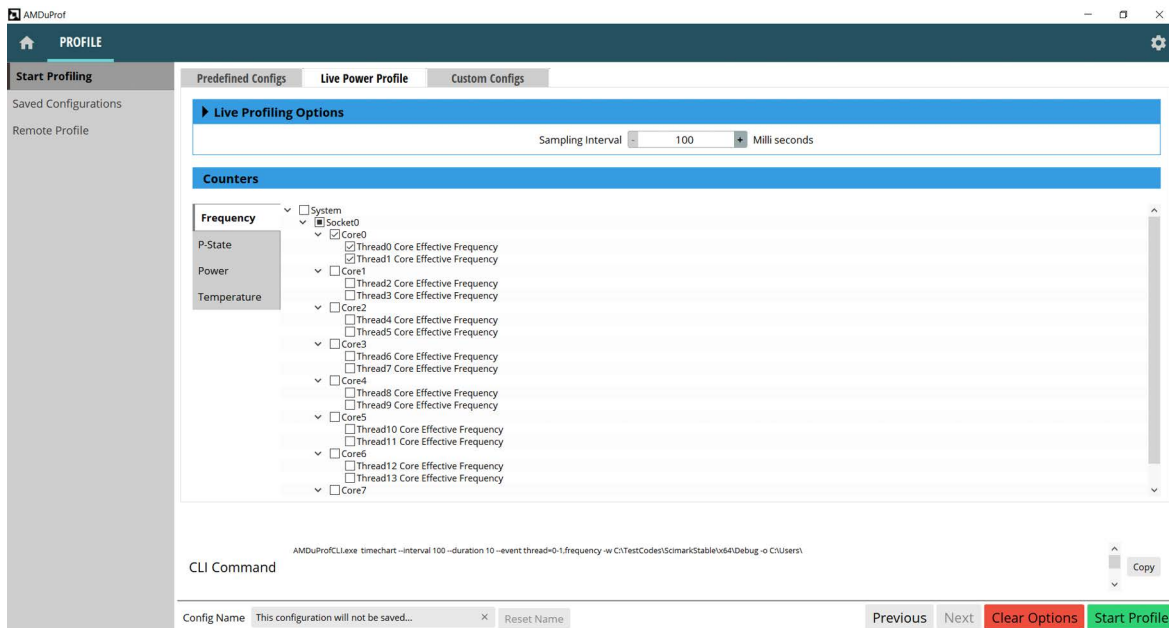


Figure 73. Live System-wide Power Profile

- 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.

The CLI Command will be displayed for all the options selected from the GUI for Live Power Profiling.

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 74. 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 75. --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 76. 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 Remote Profiling

10.1 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.

10.2 Setting up Authorization

Complete the following steps to set up the authorization:

1. Navigating to **PROFILE > Remote Profile** and locate **Client ID**:

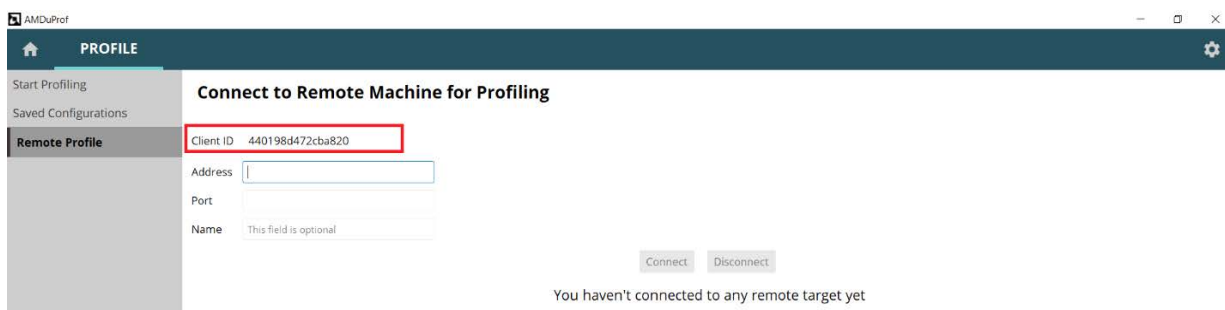


Figure 77. Client ID

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>
```

10.3 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 61. 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 78. 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 79. Selecting IP

10.4 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:

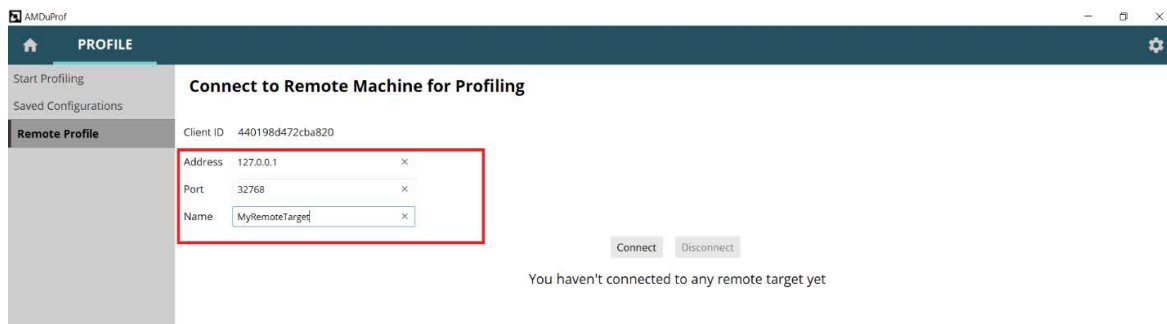


Figure 80. Connect to Remote Machine

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:

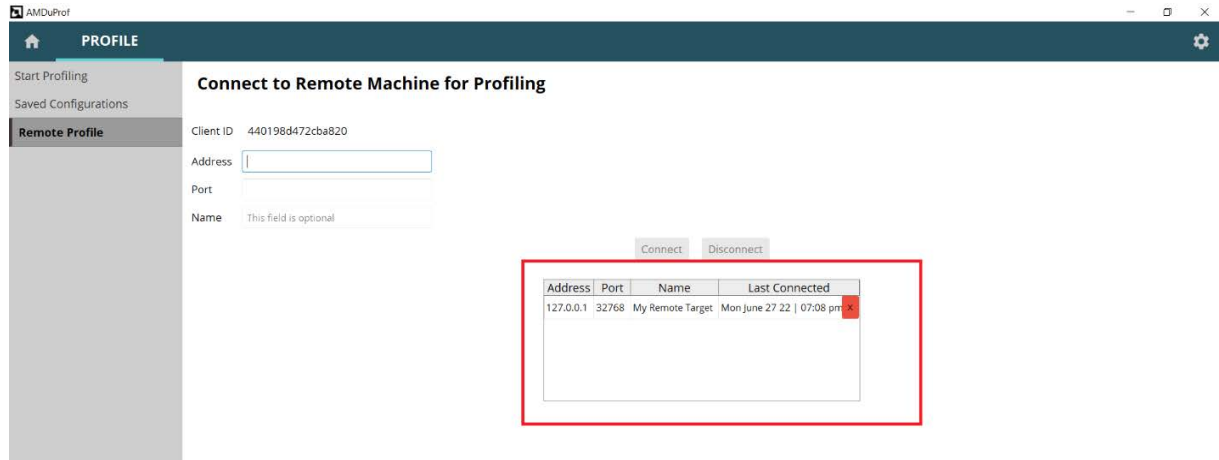


Figure 81. Remote Target Data

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:

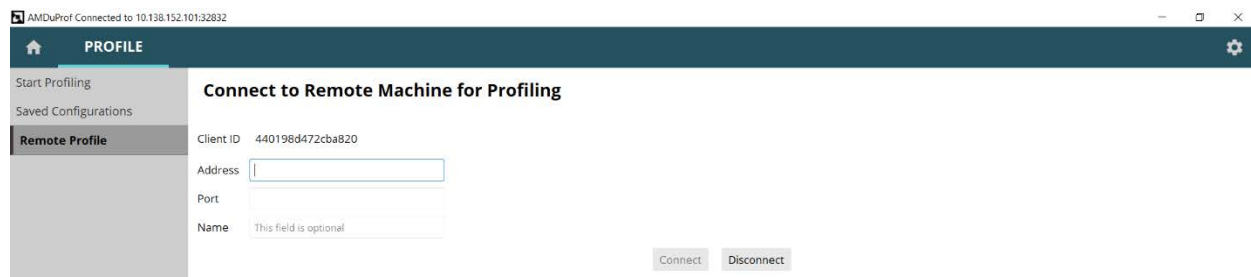


Figure 82. Disconnect Button

10.5 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 11 AMD uProf Virtualization Support

11.1 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 62. 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									
Time 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

Table 62. 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
Power Profiling									
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.

11.2 CPU Profiling

CPU Profiling supports:

- Profiling of guest VM from guest VM.
- Profiling of guest VM from host system (KVM hypervisor).

11.2.1 Profiling of Guest VM from Guest VM

Time 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.

11.2.2 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

11.2.3 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.

11.2.4 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 63. 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 63. 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.

11.2.5 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
```

11.3 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]
Core 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
```

11.4 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 12 Profile Control APIs

12.1 AMDProfileControl APIs

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

AMDProfileControl APIs work only with AMDuProfCLI and GUI for application analysis. They do not work with:

- Power Profiler
- System analysis tools (uProfPcm and uProfSys)

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
```

12.1.1 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 ();
```

amdProfilePause

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

```
bool amdProfilePause ();
```

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

12.1.2 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 collection
    amdProfileResume ();

    // Multiply the matrices
    multiply_matrices ();

    // Stop the data collection
    amdProfilePause ();

    return 0;
}
```

12.1.3 Compiling Instrumented Target Application

Windows

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

Linux

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

```
$ g++ -std=c++11 -g <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++.

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

```
$ gcc -g <sourcefile.c> -I <AMDuProf-install-dir>/include -L<AMDuProf-install-dir>/lib/x64/ -lAMDProfileController -lrt -pthread
```

12.1.4 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/cpuprof-tbp /tmp/AMDuProf/Examples/ClassicCpuProfileCtrl/ClassicCpuProfileCtrl
```

12.1.5 Limitations

The CPU profile control APIs are not supported for the MPI applications.

Chapter 13 Reference

13.1 Preparing an Application for Profiling

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

13.1.1 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**.

- From the drop-down, select **Program Database (/ZI)** or **Program Database for Edit & Continue (/ZL)**.

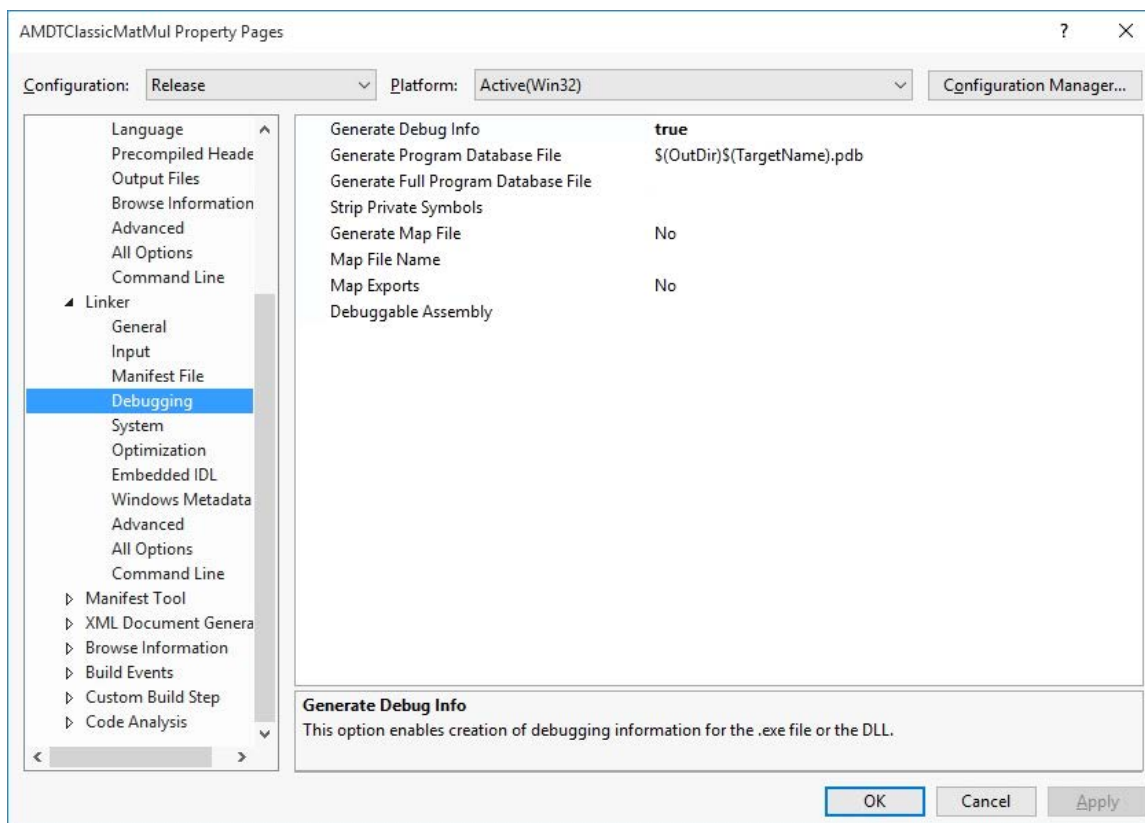


Figure 83. AMDTClassicMatMul Property Page

- In the project pane, expand **Linker** and then select **Debugging**.
- From the **Generate Debug Info** drop-down, select **/DEBUG**.

13.1.2 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.

13.2 CPU Profiling

The AMD uProf CPU Performance Profiling follows a sampling-based approach to gather the profile data periodically. It uses a variety of 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.

13.2.1 Hardware Sources

Performance Monitor Counters (PMC)

AMD 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.

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 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.

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.

13.2.2 Profiling Concepts

Sampling

Sampling profilers works based on the logic that the part of a program that consumes most of the time (or that triggers the most occurrence of the sampling event) have a larger number of samples. This is because they have a higher probability of being executed while samples are being taken by the CPU Profiler.

Sampling Interval

The time between the collection of every two samples is the Sampling Interval. For example, in TBP, if the time interval is 1 millisecond, then roughly 1,000 TBP samples are being collected every second for each processor core.

The 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.

13.2.3 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.

13.2.4 Predefined Core PMC Events

Some of the Core Performance events of AMD “Zen” processors are listed in the following table:

Table 64. 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 64. 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_MIS S	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, 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 64. 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.
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.

Table 64. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
0x47, 0x00	MISALIGNED_LOADS	<p>Misaligned Loads</p> <p>The number of misaligned loads.</p> <p><i>Note: On AMD “Zen 3” core processors, this event counts the 64B (cache-line crossing) and 4K (page crossing) misaligned loads.</i></p>
0x52, 0x03	INEFFECTIVE_SW_PF	<p>Ineffective Software Prefetches</p> <p>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.</p>
AMD EPYC™ 3rd Generation Processors		
0x76, 0x00	CYCLES_NOT_IN_HALT	<p>CPU clock cycles not halted</p> <p>The number of CPUcycles when the thread is not in halt state.</p>
0xC0, 0x00	RETIRED_INST	<p>Retired Instructions</p> <p>The number of instructions retired from execution. This count includes exceptions and interrupts. Each exception or interrupt is counted as one instruction.</p>
0xC1, 0x00	RETIRED_MACRO_OPS	<p>Retired Macro Operations</p> <p>The number of macro-ops retired. This count includes all processor activity - instructions, exceptions, interrupts, microcode assists, and so on.</p>
0xC2, 0x00	RETIRED_BR_INST	<p>Retired Branch Instructions</p> <p>The number of branch instructions retired. This includes all types of architectural control flow changes, including exceptions and interrupts</p>
0xC3, 0x00	RETIRED_BR_INST_MISP	<p>Retired Branch Instructions Mis-predicted</p> <p>The number of retired branch instructions, that were mis-predicted. Note that only EX direct mis-predicts and indirect target mis-predicts are counted.</p>

Table 64. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
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_MIS S	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.
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.

Table 64. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
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.
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.

Table 64. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
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_EXTERNAL_CACHE	L1 DC fills from local external CCX caches The Data Cache (DC) fills from cache of different CCX in the same or different package (node).
0x44, 0x48	L1_DC_REFILLS_DRAM	L1 DC fills from local Memory The Data Cache (DC) fills from DRAM or IO connected in the same or different package (node).
0x44, 0x50	L1_DC_REFILLS_REMOTE_NODE	L1 DC fills from remote node The Data Cache (DC) fills from cache of CCX in the different package (node) or the DRAM / IO connected in the different package (node).
0x44, 0x03	L1_DC_REFILLS_LOCAL_CACHE_L2_L3	L1 DC fills from same CCX The Data Cache (DC) fills from local L2 cache to the core or different L2 cache in the same CCX or L3 cache that belongs to the same CCX
0x44, 0x5F	L1_DC_REFILLS_ALL	L1 DC fills from all the data sources The Data Cache fills from all the data sources
0x60, 0xFF	L2_REQUESTS_ALL	All L2 cache requests.

Table 64. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
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.
AMD EPYC™ 4th Generation 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.

Table 64. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
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
0xC3, 0x00	RETIRED_BR_INST_MISP	Retired Branch Instructions Mis-predicted 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, 0x1F	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 counts both the 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 the LS unit. This counts the dispatch of a single op that performs a: <ul style="list-style-type: none"> • memory load • memory store • 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 the 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 the L1 data cache misses. This also counts the hardware and software prefetches.
0x64, 0x01	L2_CACHE_MISS_FROM_L1_IC_MISSES	L2 cache miss from L1 IC miss Counts all the instruction cache fill request misses in the L2 cache.

Table 64. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
0x64, 0x08	L2_CACHE_MISS_FROM_L1_DC_MIS S	L2 cache miss from L1 DC miss Counts all the data cache fill request misses in L2 cache.
0x71, 0xFF	L2_HWPF_HIT_IN_L3	L2 Prefetcher Hits in L3 Counts all the 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 the L2 prefetches accepted by the L2 pipeline which miss the L2 and 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 hit the 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 hit the L2 cache.
0x70, 0xFF	L2_HWPF_HIT_IN_L2	L2 cache hit from L2 HW Prefetch Counts all the L2 prefetches accepted by L2 pipeline which hit the L2 cache.
0x43, 0x01	L1_DEMAND_DC_REFILLS_LOCAL_ L2	L1 demand DC fills from L2 The demand Data Cache (DC) fills from the 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_DEMAND_DC_REFILLS_EXTERN AL_CACHE_LOCAL	L1 DC fills from local external CCX caches The DC fills from the 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 DC fills from DRAM or IO connected in the same package (node).
0x43, 0x10	L1_DEMAND_DC_REFILLS_EXTERN AL_CACHE_REMOTE	L1 demand DC fills from remote external cache The demand DC fills from the CCX cache in the different package (node).

Table 64. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
0x43, 0x40	L1_DEMAND_DC_REFILLS_REMOTE_DRAM	L1 demand DC fills from remote Memory The demand 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 DC fills from the cache of different CCX in the same or different package (node).
0x43, 0xDF	L1_DEMAND_DC_REFILLS_ALL	L1 demand DC refills from all data sources. The demand DC fills from all the data sources.
0x44, 0x01	L1_DC_REFILLS_LOCAL_L2	L1 DC fills from local L2 The DC fills from the local L2 cache to the core.
0x44, 0x02	L1_DC_REFILLS_LOCAL_CACHE	L1 DC fills from local CCX cache The DC fills from different L2 cache in the same CCX or L3 cache that belongs to the same CCX.
0x44, 0x08	L1_DC_REFILLS_EXTERNAL_CACHE_LOCAL	L1 DC fills from local Memory The 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 DC fills from the 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 DC fills from the CCX cache in the different package (node).
0x44, 0x40	L1_DC_REFILLS_REMOTE_DRAM	L1 DC fills from remote Memory The DC fills from DRAM or IO connected in the different package (node).
0x44, 0x14	L1_DC_REFILLS_EXTERNAL_CACHE	L1 DC fills from local external CCX caches The 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 DC fills from DRAM or IO connected in the same or different package (node).

Table 64. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
0x44, 0x50	L1_DC_REFILLS_REMOTE_NODE	L1 DC fills from remote node The DC fills from the CCX cache 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 DC fills from the local L2 cache to the core or different L2 cache in the same CCX or L3 cache that belongs to the same CCX.
0x44, 0xDF	L1_DC_REFILLS_ALL	L1 DC fills from all the data sources The DC fills from all the data sources
0x60, 0xFF	L2_REQUESTS_ALL	All L2 cache requests.
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 64 B (cache-line crossing) and 4 K (page crossing) misaligned loads.</i>

Table 64. Predefined Core PMC Events

Event Id, Unit-mask	Event Abbreviation	Name and Description
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 allocated miss request buffer. Also counts the Software PREFETCH instruction that saw a DC hit.
0x18E, 0x1F	IC_TAG_ALL_IC_ACCESS	IC Tag All Instruction Cache Access
0x18E, 0x18	IC_TAG_IC_MISS	IC Tag Instruction Cache Miss
0x28F, 0x07	OP_CACHE_ALL_ACCESS	All OP Cache Accesses
0x28F, 0x04	OP_CACHE_MISS	Op Cache Miss

Following table shows the CPU performance metrics:

Table 65. 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.

Table 65. Core CPU Metrics

CPU Metric	Description
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.
RETIRED_BR_INST_MISP_RATIO	This metric is computed as the retired mis-predicted branches divided by the total number of retired branch instructions.
RETIRED_BR_INST_RATE	The number of retired branch instructions rate. This metric is computed as the retired branches divided by the total number of retired instructions.
RETIRED_BR_INST_MISP_RATE	This metric is computed as retired mis-predicted branches divided by the total number of retired instructions.
RETIRED_TAKEN_BR_INST (PTI)	The number of retired taken branches per thousand instructions.
RETIRED_TAKEN_BR_INST_RATE	The number of retired taken branches rate. This metric is computed as the retired taken branches divided by the total number of retired instructions.
RETIRED_TAKEN_BR_INST_MISP (PTI)	The number of retired mis-predicted taken branches per thousand instructions.
RETIRED_INDIRECT_BR_INST_MISP (PTI)	The number of retired indirect branches per thousand instructions.
RETIRED_NEAR_RETURNS (PTI)	The number of retired near branches per thousand instructions.
RETIRED_NEAR_RETURNS_MISP (PTI)	The number of retired mis-predicted near branches per thousand instructions.
RETIRED_NEAR_RETURNS_MISP_RATE	This metric is computed as the retired mis-predicted near returns divided by the total number of retired instructions.
RETIRED_NEAR_RETURNS_MISP_RATIO	This metric is computed as retired mis-predicted near returns divided by the total number of retired return instructions.
L1_DTLB_MISS_RATE	The DTLB L1 miss rate is the number of DTLB L1 misses divided by the total number of retired instructions.
L2_DTLB_MISS_RATE	The L2 DTLB miss rate is the number of L2 DTLB misses divided by the total number of retired instructions.

Table 65. Core CPU Metrics

CPU Metric	Description
L1_ITLB_MISS_RATE	The ITLB L1 miss rate is the number of ITLB L1_Miss_L2_Hits and L1_Miss_L2_Miss divided by the total number of retired instructions.
L2_ITLB_MISS_RATE	The ITLB L2 miss rate is the number of ITLB L2 miss divided by the total number of retired instructions.
MISALIGNED_LOADS_RATIO	The misalign ratio is the number of misaligned loads divided by the total number of DC accesses.
MISALIGNED_LOADS_RATE	The misalign rate is the number of misaligned loads divided by the total number of retired instructions.
STLI_OTHER	Store-to-load conflicts: A load was unable to complete due to a non-forwardable conflict with an older store. Most commonly, a load's address range partially but not completely overlaps with an uncompleted older store. Software can avoid this problem by using the same size and alignment loads and stores when accessing the data. Vector/SIMD code is particularly susceptible to this problem; software should construct wide vector stores by manipulating the vector elements in the registers using shuffle/blend/swap instructions prior to storing to the memory, instead of using narrow element-by-element stores.
L2_CACHE_ACCESSES_FROM_IC_MISSES	The number of L2 cache access requests due to the L1 instruction cache misses per thousand retired instructions. This L2 cache access requests also includes the prefetches.
L2_CACHE_MISSES_FROM_IC_MISSES	The number of L2 cache misses from L1 instruction cache misses per thousand retired instructions.

13.2.5 IBS Derived Events

AMD uProf translates the IBS information produced by the hardware into derived event sample counts that resemble EBP sample counts. All 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.

Following table shows the IBS fetch events:

Table 66. IBS Fetch Events

IBS Fetch Event	Description
AMD “Zen1”, AMD “Zen2”, and AMD “Zen3” Client Platforms	
IBS_FETCH	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_ATTEMPT	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_COMP	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_ABORT	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_L1_ITLB_HIT	The number of IBS attempted fetch samples where the fetch operation initially hit in the L1 ITLB (Instruction Translation Lookaside Buffer).
IBS_ITLB_L1M_L2H	The number of IBS attempted fetch samples where the fetch operation initially missed in the L1 ITLB and hit in the L2 ITLB.
IBS_ITLB_L1M_L2M	The number of IBS attempted fetch samples where the fetch operation initially missed in both the L1 ITLB and the L2 ITLB.
IBS_IC_MISS	The number of IBS attempted fetch samples where the fetch operation initially missed in the IC (instruction cache).
IBS_IC_HIT	The number of IBS attempted fetch samples where the fetch operation initially hit in the IC.

Table 66. IBS Fetch Events

IBS Fetch Event	Description
IBS_4K_PAGE	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.
IBS_2M_PAGE	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_LAT	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_L2C_MISS	The instruction fetch missed in the L2 Cache.
IBS_ITLB_REFILL_LAT	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.
AMD “Zen3” and AMD “Zen4” Server Platforms	
IBS_FETCH	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_ATTEMPTED	The number of IBS sampled fetches that were not killed fetch attempts. This derived event measures the number of useful fetch attempts and does not include the number of IBS killed fetch samples. This event should be used to compute ratios such as the ratio of IBS fetch IC misses to attempted fetches. The number of attempted fetches should equal the sum of the number of completed fetches and the number of aborted fetches.
IBS_FETCH_COMPLETED	The number of IBS sampled fetches that completed. A fetch is completed if the attempted fetch delivers instruction data to the instruction decoder. Although the instruction data was delivered, it may still not be used (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 does 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 mis-predicted branch. The number of IBS aborted fetch samples is a lower bound on the amount of unsuccessful, speculative fetch activity. It is a lower bound as the instruction data delivered by completed fetches may not be used.
IBS_FETCH_L1_ITLB_HIT	The number of IBS attempted fetch samples where the fetch operation initially hit in the L1 ITLB (Instruction Translation Lookaside Buffer).
IBS_FETCH_L1_ITLB_MISS_L2_ITLB_HIT	The number of IBS attempted fetch samples where the fetch operation initially missed in the L1 ITLB and hit in the L2 ITLB.
IBS_FETCH_L1_ITLB_MISS_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.

Table 66. IBS Fetch Events

IBS Fetch Event	Description
IBS_FETCH_L1_IC_MIS S	The number of IBS attempted fetch samples where the fetch operation initially missed in the IC (instruction cache).
IBS_FETCH_L1_IC_HIT	The number of IBS attempted fetch samples where the fetch operation initially hit in the IC.
IBS_FETCH_L1_ITLB_4 K_PAGE	The number of IBS attempted fetch samples where the fetch operation produced a valid physical address (for example, address translation completed successfully) and used a 4 KB page entry in the L1 ITLB.
IBS_FETCH_L1_ITLB_2 M_PAGE	The number of IBS attempted fetch samples where the fetch operation produced a valid physical address (for example, address translation completed successfully) and used a 2 MB page entry in the L1 ITLB.
IBS_FETCH_L1_ITLB_1 G_PAGE	The number of IBS attempted fetch samples where the fetch operation produced a valid physical address (for example, address translation completed successfully) and used a 1 GB page entry in the L1 ITLB.
IBS_FETCH_LAT	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_MISS	The instruction fetch missed in the L2 Cache.
IBS_FETCH_ITLB_REFI LL_LAT	The number of cycles when the fetch engine is stalled for an ITLB reload for the sampled fetch. If there is no reload, the latency will be 0.
IBS_FETCH_OP_CACH E_MISS	The number of IBS attempted fetch samples where the Op Cache was not able to supply all the bytes for the tagged fetch.
IBS_FETCH_L3_MISS	The number of IBS attempted fetch samples where the instruction fetch missed in the L3 cache on the same CCX.

Following table lists the IBS fetch metrics:

Table 67. IBS Fetch Metrics

IBS Fetch Metric	Description
IBS_FETCH_L1_IC_MISS_R ATE_%	Percentage of IBS fetch L1 instruction cache misses with respect to the total number of IBS fetch attempts.
IBS_FETCH_LAT_AVE	The average IBS fetch latency. Calculated by dividing the IBS fetch latency by the total number of IBS fetch attempts.
IBS_FETCH_L1_ITLB_MISS_ L2_ITLB_HIT_RATE_%	Percentage of IBS fetch L1 ITLB miss and L2 ITLB hits with respect to the total number of IBS fetch attempts.
IBS_FETCH_L1_ITLB_MISS_ L2_ITLB_MISS_RATE_%	Percentage of IBS fetch L1 and L2 ITLB misses with respect to the total number of IBS fetch attempts.

Following table lists the IBS op events:

Table 68. IBS Op Events

IBS Op Event	Description
AMD “Zen1”, “Zen2”, and “Zen3” Client Platforms	
IBS_ALL_OPS	The number of all the IBS op samples collected. These op samples may be branch ops, resync ops, ops that perform load/store operations, or undifferentiated ops (for example, those ops that perform arithmetic operations, logical operations, and so on). IBS collects data for the retired ops. No data is collected for the ops that are aborted due to pipeline flushes and so on. Thus, all the sampled ops are architecturally significant and contribute to the successful execution of programs.
IBS_TAG_TO_RET	The total number of tag-to-rotate cycles across all the IBS op samples. The tag-to-rotate time of an op is the number of cycles from when the op was tagged (selected for sampling) to when the op retired.
IBS_COMP_TO_RET	The total number of completion-to-rotate cycles across all the IBS op samples. The completion-to-rotate time of an op is the number of cycles from when the op completed to when the op retired.
IBS_BR	The number of IBS retired branch op samples. A branch operation is a change in the program control flow and includes unconditional and conditional branches, subroutine calls, and subroutine returns. Branch ops are used to implement AMD64 branch semantics.
IBS_MISP_BR	The number of IBS samples for retired branch operations that were mis-predicted. This event should be used to compute the ratio of mis-predicted branch operations to all the branch operations.
IBS_TAKEN_BR	The number of IBS samples for the retired branch operations that were taken branches.
IBS_MISP_TAKEN_BR	The number of IBS samples for the retired branch operations that were mis-predicted taken branches.
IBS_RET	The number of IBS retired branch op samples where the operation was a subroutine return. These samples are a subset of all the IBS retired branch op samples.
IBS_MISP_RET	The number of IBS retired branch op samples where the operation was a mis-predicted subroutine return. This event should be used to compute the ratio of the mis-predicted returns to all the subroutine returns.
IBS_RESYNC	The number of IBS resync op samples. A resync op is only found in certain microcoded AMD64 instructions and causes a complete pipeline flush.
IBS_LOAD_STORE	The number of IBS op samples for ops that perform either a load and/or store operation. Each op may perform a load operation, a store operation, or both a load and store operation (each to the same address).
IBS_LOAD	The number of IBS op samples for ops that perform a load operation.

Table 68. IBS Op Events

IBS Op Event	Description
IBS_STORE	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 the L1 DTLB (data translation lookaside buffer).
IBS_DTLB_L1M_L2H	The number of IBS op samples where either a load or store operation initially missed in the L1 DTLB and hit the L2 DTLB.
IBS_DTLB_L1M_L2M	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_DC_MISS	The number of IBS op samples where either a load or store operation initially missed in the L1 DC.
IBS_DC_HIT	The number of IBS op samples where either a load or store operation initially hit the L1 DC.
IBS_MISALIGN_ACC	The number of IBS op samples where either a load or store operation caused a misaligned access (for example, the load or store operation crossed a 128-bit boundary).
IBS_BANK_CONF_LOAD	The number of IBS op samples where either a load or store operation caused a bank conflict with a load operation.
IBS_BANK_CONF_STORE	The number of IBS op samples where either a load or store operation caused a bank conflict with a store operation.
IBS_FORWARDED	The number of IBS op samples where data for a load operation was forwarded from a store operation.
IBS_STLF_CANCELLED	The number of IBS op samples where data forwarding to a load operation from a store was canceled.
IBS_UC_MEM_ACC	The number of IBS op samples where a load or store operation accessed uncacheable (UC) memory.
IBS_WC_MEM_ACC	The number of IBS op samples where a load or store operation accessed write combining (WC) memory.
IBS_LOCKED_OP	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	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 the address translation.
IBS_L1_DTLB_2M	The number of IBS op samples where a load or store operation produced a valid linear (virtual) address and a 2 M page entry in the L1 DTLB was used for the address translation.

Table 68. IBS Op Events

IBS Op Event	Description
IBS_L1_DTLB_1G	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 the address translation.
IBS_L2_DTLB_4K	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 the address translation.
IBS_L2_DTLB_2M	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 the address translation.
IBS_L2_DTLB_1G	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_LOAD_DC_MISS_LAT	The total L1 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 L1 data cache miss was detected to when data was delivered to the core.
IBS_LOAD_RESYNC	Load Resync.
IBS_NB_LOCAL	The number of IBS op samples where a load operation was serviced from the local processor. Northbridge IBS data is only valid for the load operations that miss in both the L1 data cache and the L2 data cache. If a load operation crosses a cache line boundary, the IBS data reflects the access to the lower cache line.
IBS_NB_REMOTE	The number of IBS op samples where a load operation was serviced from a remote processor.
IBS_NB_LOCAL_L3	The number of IBS op samples where a load operation was serviced by the local L3 cache.
IBS_NB_LOCAL_CACHE	The number of IBS op samples where a load operation was serviced by a cache (L1 or L2 data cache) belonging to a local core which is a sibling of the core making the memory request.
IBS_NB_REMOTE_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 Hyper Transport links.
IBS_NB_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_NB_REMOTE_DRAM	The number of IBS op samples where a load operation was serviced by the remote system memory (after traversing one or more coherent Hyper Transport links and through a remote memory controller).
IBS_NB_LOCAL_OTHER	The number of IBS op samples where a load operation was serviced from local MMIO, configuration or PCI space, or from the local APIC.

Table 68. IBS Op Events

IBS Op Event	Description
IBS_NB_REMOTE_OTHER	The number of IBS op samples where a load operation was serviced from remote MMIO, configuration, or PCI space.
IBS_NB_CACHE_MODIFIED	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_NB_CACHE_OWNED	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_NB_LOCAL_LAT	The total data cache miss latency (in processor cycles) for the load operations that were serviced by the local processor.
IBS_NB_REMOTE_LAT	The total data cache miss latency (in processor cycles) for the load operations that were serviced by a remote processor.
AMD “Zen4” and AMD “Zen3” Server Platforms	
IBS_ALL_OPS	The number of all the IBS op samples that were collected. These samples may be branch ops, resync ops, ops that perform load/store operations, or undifferentiated ops. For example, the ops that perform arithmetic operations, logical operations, and so on. IBS collects data for retired ops. No data is collected for ops that are aborted due to pipeline flushes and so on. Thus, all sampled ops are architecturally significant and contribute to the successful program execution.
IBS_TAG_TO_RET	The total number of tag-to-rotate cycles across all the IBS op samples. The tag-to-rotate time of an op is the number of cycles from when the op was tagged (selected for sampling) to when the op retired.
IBS_COMP_TO_RET	The total number of completion-to-rotate cycles across all the IBS op samples. The completion-to-rotate time of an op is the number of cycles from when the op completed to when the op retired.
IBS_BR	The number of IBS retired branch op samples. A branch operation is a change in program control flow; includes unconditional and conditional branches, subroutine and subroutine returns. Branch ops are used to implement AMD64 branch semantics.
IBS_MISP_BR	The number of IBS samples for the retired branch operations that were mis-predicted. This event should be used to compute the ratio of mis-predicted branch operations to all branch operations.
IBS_TAKEN_BR	The number of IBS samples for retired branch operations that were taken branches.
IBS_MISP_TAKEN_BR	The number of IBS samples for the retired branch operations that were mis-predicted taken branches.
IBS_RET	The number of IBS retired branch op samples where the operation was a subroutine return. These samples are a subset of all the IBS retired branch op samples.

Table 68. IBS Op Events

IBS Op Event	Description
IBS_MISP_RET	The number of IBS retired branch op samples where the operation was a mis-predicted subroutine return. This event should be used to compute the ratio of the mis-predicted returns to all the subroutine returns.
IBS_RESYNC	The number of IBS resync op samples. A resync op is only found in certain microcoded AMD64 instructions and causes a complete pipeline flush.
IBS_LOAD_STORE	The number of IBS op samples for the ops that perform either a load and/or store operation. Each op may perform a load/store operation or both a load and store operation (each to the same address).
IBS_LOAD	The number of IBS op samples for the ops that perform a load operation.
IBS_STORE	The number of IBS op samples for the 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 look aside buffer).
IBS_DTLB_L1M_L2H	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_DTLB_L1M_L2M	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_DC_MISS	The number of IBS op samples where either a load or store operation initially missed in the L1 data cache (DC).
IBS_DC_HIT	The number of IBS op samples where either a load or store operation initially hit in the L1 data cache (DC).
IBS_MISALIGN_ACC	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_CONF_LOAD	The number of IBS op samples where either a load or store operation caused a bank conflict with a load operation.
IBS_BANK_CONF_STORE	The number of IBS op samples where either a load or store operation caused a bank conflict with a store operation.
IBS_FORWARDED	The number of IBS op samples where data for a load operation was forwarded from a store operation.
IBS_STLF_CANCELLED	The number of IBS op samples where data forwarding to a load operation from a store was canceled.
IBS_UC_MEM_ACC	The number of IBS op samples where a load or store operation accessed uncacheable (UC) memory.
IBS_WC_MEM_ACC	The number of IBS op samples where a load or store operation accessed write combining (WC) memory.
IBS_LOCKED_OP	The number of IBS op samples where a load or store operation was a locked operation.

Table 68. IBS Op Events

IBS Op Event	Description
IBS_MAB_HIT	The number of IBS op samples where a load or store operation hit an allocated entry in the Miss Address Buffer (MAB).
IBS_L1_DTLB_4K	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 L1 DTLB was used for the address translation.
IBS_L1_DTLB_2M	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 L1 DTLB was used for the address translation.
IBS_L1_DTLB_1G	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 L1 DTLB was used for the address translation.
IBS_L2_DTLB_4K	The number of IBS op samples where a load or store operation produced a valid linear (virtual) address, hit L2 DTLB, and used a 4 KB page entry for the address translation.
IBS_L2_DTLB_2M	The number of IBS op samples where a load or store operation produced a valid linear (virtual) address, hit L2 DTLB, and used a 2 MB page entry for the address translation.
IBS_L2_DTLB_1G	The number of IBS op samples where a load or store operation produced a valid linear (virtual) address, hit L2 DTLB, and used a 1 GB page entry for the address translation.
IBS_LOAD_DC_MISS_LAT	The total L1 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 L1 data cache miss was detected to when data was delivered to the core.
IBS_LOAD_RESYNC	Load Resync.
IBS_NB_LOCAL	The number of IBS op samples where a load operation was serviced from the local processor. Northbridge IBS data is only valid for the load operations that miss in both the L1 and L2 data cache. If a load operation crosses a cache line boundary, the IBS data reflects the access to the lower cache line.
IBS_NB_REMOTE	The number of IBS op samples where a load operation was serviced from a remote processor.
IBS_NB_LOCAL_L3	The number of IBS op samples where a load operation was serviced by the local L3 cache.
IBS_NB_LOCAL_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 that is a sibling of the core making the memory request.

Table 68. IBS Op Events

IBS Op Event	Description
IBS_NB_REMOTE_CACHE	The number of IBS op samples where a load operation was serviced by a remote L1 data, L2, or L3 cache after traversing one or more coherent HyperTransport links.
IBS_NB_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_NB_REMOTE_DRAM	The number of IBS op samples where a load operation was serviced by the remote system memory (after traversing one or more coherent HyperTransport links and through a remote memory controller).
IBS_NB_LOCAL_OTHER	The number of IBS op samples where a load operation was serviced from the local MMIO, configuration, PCI space, or local APIC.
IBS_NB_REMOTE_OTHER	The number of IBS op samples where a load operation was serviced from the remote MMIO, configuration, or PCI space.
IBS_NB_CACHE_MODIFIED	The number of IBS op samples where a load operation was serviced from the local or remote cache, and the cache hit state was the Modified (M) state.
IBS_NB_CACHE_OWNED	The number of IBS op samples where a load operation was serviced from the local or remote cache, and the cache hit state was the Owned (O) state.
IBS_NB_LOCAL_LAT	The total data cache miss latency (in processor cycles) for the load operations that were serviced by the local processor.
IBS_NB_REMOTE_LAT	The total data cache miss latency (in processor cycles) for the load operations that were serviced by a remote processor.

Following table lists the IBS op metrics for AMD “Zen4” and AMD “Zen3” server platforms:

Table 69. IBS Op Metrics for AMD “Zen4” and AMD “Zen3” Server Platforms

IBS Op Metric	Description
%IBS_BR_TAG_TO_RETIRE_CYCLES	Percentage of IBS Branch op tag to retire cycles.
%IBS_BR_MISP_TAG_TO_RETIRE_CYCLES	Percentage of IBS Branch mis-predict op tag to retire cycles.
%IBS_TAKEN_BR_TAG_TO_RETIRE_CYCLES	Percentage of IBS Branch taken op tag to retire cycles.
%IBS_RET_TAG_TO_RETIRE_CYCLES	Percentage of IBS Branch return op tag to retire cycles.
%IBS_BR_COMP_TO_RETIRE_CYCLES	Percentage of IBS Branch op completion to retire cycles.
%IBS_BR_MISP_COMP_TO_RETIRE_CYCLES	Percentage of IBS Branch mis-predict op completion to retire cycles.
%IBS_TAKEN_BR_COMP_TO_RETIRE_CYCLE S	Percentage of IBS Branch taken op completion to retire cycles.

Table 69. IBS Op Metrics for AMD “Zen4” and AMD “Zen3” Server Platforms

IBS Op Metric	Description
%IBS_RET_COMP_TO_RETIRE_CYCLES	Percentage of IBS Branch return op completion to retire cycles.
IBS_BR_MISP_RATE_%	Branch mis-predict rate in percentage. The number of branch mis-predicts divided by the total number of branch operations, expressed as percentage.
%IBS_L1_DTLB_REFILL_LAT_CYCLES	Percentage of cycles wasted due to L1 DTLB misses. The number of L1 DTLB refill latency cycles divided by the total number of Tag-To-Retire cycles of all the operations, expressed as percentage.
IBS_ST_L1_DC_MISS_RATE_%	Store L1 DC Miss rate in percentage. The number of store L1 DC misses divided by the total number of store ops, expressed as percentage.
IBS_LD_L1_DC_MISS_RATE_%	Load L1 DC Miss rate in percentage. The number of load L1 DC misses divided by the total number of load ops, expressed as percentage.
IBS_LD_L1_DC_HIT_RATE_%	Load L1 DC Hit rate in percentage. The number of load L1 DC hits divided by the total number of load ops, expressed as percentage.
IBS_LD_L2_HIT_RATE_%	Load L2 Hit rate in percentage. The number of load L2 hits divided by the total number of load ops, expressed as percentage.
IBS_LD_LOCAL_CACHE_HIT_RATE_%	Percentage of load samples where the load operation was serviced by the shared L3 cache or other L1/L2 cache in the same CCX. The number of IBS_LD_LOCAL_CACHE_HIT divided by IBS_LOAD, expressed in percentage.
IBS_LD_PEER_CACHE_HIT_RATE_%	Percentage of load samples where the load operation was serviced by L2/L3 cache in a different CCX of same NUMA node. The number of IBS_LD_PEER_CACHE_HIT divided by IBS_LOAD, expressed in percentage.
IBS_LD_RMT_CACHE_HIT_RATE_%	Percentage of load samples where the load operation was serviced by L2/L3 cache of different CCX in different NUMA node. The number of IBS_LD_RMT_CACHE_HIT divided by IBS_LOAD, expressed in percentage.

Table 69. IBS Op Metrics for AMD “Zen4” and AMD “Zen3” Server Platforms

IBS Op Metric	Description
IBS_LD_LOCAL_DRAM_HIT_RATE_%	Percentage of load samples where the load operation was serviced by local system memory (local DRAM via the memory controller) of same NUMA node. The number of IBS_LD_LOCAL_DRAM_HIT divided by IBS_LOAD, expressed in percentage.
IBS_LD_RMT_DRAM_HIT_RATE_%	Percentage of load samples where the load operation was serviced by DRAM in different NUMA node. The number of IBS_LD_RMT_DRAM_HIT divided by IBS_LOAD, expressed in percentage.
IBS_LD_DRAM_HIT_RATE_%	Percentage of load samples where the load operation was serviced by DRAM in the system. The number of IBS_LD_DRAM_HIT divided by IBS_LOAD, expressed in percentage.
IBS_LD_NVDIMM_HIT_RATE_%	Percentage of load samples where the load operation was serviced by NVDIMM in the system. The number of IBS_LD_NVDIMM_HIT divided by IBS_LOAD, expressed in percentage.
IBS_LD_EXT_MEM_HIT_RATE_%	Percentage of load samples where the load operation was serviced by Extension Memory in the system. The number of IBS_LD_EXT_MEM_HIT divided by IBS_LOAD, expressed in percentage.
IBS_LD_PEER_AGENT_MEM_RATE_%	Percentage of load samples where the load operation was serviced by Peer agent Memory in the system. The number of IBS_LD_EXT_MEM_HIT divided by IBS_LOAD, expressed in percentage.
IBS_LD_NON_MAIN_MEM_HIT_RATE_%	Percentage of load samples where the load operation was serviced from MMIO, configuration or PCI space, or from the local APIC in the system. The number of IBS_LD_NON_MAIN_MEM_HIT divided by IBS_LOAD, expressed in percentage.
IBS_LD_L1_DC_MISS_LAT_AVE	Average Load L1 DC Miss latency cycles. The number of load L1 DC misses latency divided by the total number of load L1 DC misses latency cycles.
%IBS_LD_L1_DC_MISS_LAT_CYCLES	Percentage of cycles wasted to fetch the data. The number of Load L1 DC misses latency cycles divided by the total number of Tag-To-Retire cycles of all the operations, expressed as percentage.
%IBS_LD_L2_HIT_LAT	Percentage of IBS load L2 hit latency cycles wrt. load L1 DC miss latency cycles.

Table 69. IBS Op Metrics for AMD “Zen4” and AMD “Zen3” Server Platforms

IBS Op Metric	Description
%IBS_LD_LOCAL_CACHE_HIT_LAT	Percentage of IBS load local cache hit latency cycles with respect to the load L1 DC miss latency cycles.
%IBS_LD_PEER_CACHE_HIT_LAT	Percentage of IBS load peer cache hit latency cycles with respect to the load L1 DC miss latency cycles.
%IBS_LD_RMT_CACHE_HIT_LAT	Percentage of IBS load remote cache hit latency cycles with respect to the load L1 DC miss latency cycles.
%IBS_LD_LOCAL_DRAM_HIT_LAT	Percentage of IBS load local DRAM hit latency cycles with respect to the load L1 DC miss latency cycles.
%IBS_LD_RMT_DRAM_HIT_LAT	Percentage of IBS load remote DRAM hit latency cycles with respect to the load L1 DC miss latency cycles.
%IBS_LD_DRAM_HIT_LAT	Percentage of IBS load DRAM hit latency cycles with respect to the load L1 DC miss latency cycles.
%IBS_LD_NVDIMM_HIT_LAT	Percentage of IBS load NVDIMM hit latency cycles with respect to the load L1 DC miss latency cycles.
%IBS_LD_EXTN_MEM_HIT_LAT	Percentage of IBS load Extension Memory hit latency cycles with respect to the load L1 DC miss latency cycles.
%IBS_LD_PEER_AGENT_MEM_HIT_LAT	Percentage of IBS load Peer Agent Memory hit latency cycles with respect to the load L1 DC miss latency cycles.
%IBS_LD_NON_MAIN_MEM_HIT_LAT	Percentage of IBS load Non main memory hit latency cycles with respect to the load L1 DC miss latency cycles.

13.3 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>)