



uProf ユーザー ガイド

発行番号	57368	改訂番号	4.2
発行日	2024 年 1 月		

この資料は表記のバージョンの英語版を翻訳したもので、内容に相違が生じる場合には原文を優先します。資料によっては英語版の更新に対応していないものがあります。日本語版は参考用としてご使用の上、最新情報につきましては、必ず最新英語版をご参照ください。

この資料に含まれる情報は情報提供のみを目的としており、予告なしに変更される場合があります。この資料の作成に際してはあらゆる予防措置を講じていますが、技術的に不正確な点、欠落、印刷上の間違いが含まれている可能性があります。AMD にはこの情報を更新または訂正する義務はないものとします。Advanced Micro Devices, Inc は、このドキュメントの内容の正確性や完全性について、一切の表明も保証も行いません。また、この資料に記載された AMD のハードウェア、ソフトウェア、その他の製品の操作や使用に関して、非侵害、商品性、または特定目的への適合性に関する暗示の保証を含む、いかなる責任も負わないものとします。この資料は、暗示または禁反言の法理によって生じるいかなる知的所有権への許諾を与えるものではありません。AMD の製品の購入または使用に適用される条件および制限は、当事者間で締結された契約または AMD の標準販売条件に規定されています。

商標

AMD、AMD の矢印形のロゴ、およびその組み合わせは、Advanced Micro Devices, Inc の商標です。

Dolby は Dolby Laboratories の商標です。

ENERGY STAR は U.S. Environmental Protection Agency の登録商標です。

HDMI は HDMI Licensing, LLC の商標です。

HyperTransport は、HyperTransport Technology Consortium の許諾商標です。

Microsoft、Windows、Windows Vista、DirectX は、Microsoft Corporation の登録商標です。

MMX は Intel Corporation の商標です。

OpenCL は Apple Inc. の商標であり、Khronos による許可を受けて使用されています。

PCIe は PCI-Special Interest Group (PCI-SIG) の登録商標です。

この資料で使用されているその他の製品名は識別のみを目的としたものであり、各社の商標である可能性があります。

Dolby Laboratories, Inc.

Dolby Laboratories が提供するライセンスに従って製造されています。

Rovi Corporation

このデバイスは米国特許とその他の知的財産権によって保護されています。デバイスでの Rovi Corporation のコピー保護テクノロジーの使用は、Rovi Corporation による許可を必要とし、Rovi Corporation によって書面による許可を得ている場合を除き、自宅およびその他の限定的なペーパービュー方式での使用のみを対象としています。

リバース エンジニアリングおよび分解は禁止されています。

MPEG-2 特許ポートフォリオ内の適用可能な特許に基づくライセンスなしで、MPEG-2 規格に準拠するいかなる方法においてもこの製品を使用することは明示的に禁じられています。該当ライセンスは、MPEG LA, L.L.C. (6312 S. FIDDLERS GREEN CIRCLE, SUITE 400E, GREENWOOD VILLAGE, COLORADO 80111) から提供されています。

目次

改訂履歴	16
このユーザー ガイドについて	17
対象者	17
表記規則	17
略語	18
用語	19
パート 1:	
はじめに	1
第 1 章 はじめに	2
1.1 概要	2
1.2 仕様	3
1.2.1 プロセッサ	3
1.2.2 オペレーティング システム	3
1.2.3 コンパイラとアプリケーション環境	3
1.2.4 仮想化のサポート	4
1.2.5 コンテナのサポート	4
1.3 AMD uProf のインストール	5
1.3.1 Windows	5
1.3.2 Linux	5
1.3.3 FreeBSD	7
1.4 サンプル プログラム	7
1.5 サポート	7
パート 2:	
システム解析	8
第 2 章 AMDuProfPcm の使用開始	9
2.1 概要	9
2.1.1 使用要件	9
2.2 オプション	10
2.3 コマンド	26
2.4 例	26
2.4.1 Linux と FreeBSD	26
2.4.2 Windows	27

2.5	BIOS 設定 - 既知の動作	29
2.6	ルートの権限なしでの監視	29
2.7	ルーフライン モデル	30
2.8	パイプライン使用率	32
第 3 章	AMDuProfSys の使用開始	35
3.1	概要	35
3.2	サポートされるプラットフォーム	35
3.3	サポートされるハードウェア カウンター	35
3.4	サポートされるオペレーティング システム	35
3.5	セットアップ	36
3.5.1	Linux	36
3.5.2	Windows	36
3.6	オプション	38
3.6.1	一般	38
3.6.2	収集コマンド	38
3.6.3	レポート コマンド	39
3.7	例	40
3.8	制限	40
パート 3:		
アプリケーション解析		41
第 4 章	ワークフローと主な概念	42
4.1	ワークフロー	42
4.1.1	収集フェーズ	42
4.1.2	変換およびレポート プロセス	44
4.1.3	解析フェーズ	44
4.2	事前定義サンプリング設定	44
4.3	事前定義ビュー設定	46
第 5 章	AMD uProf GUI の使用開始	50
5.1	ユーザー インターフェイス	50
5.2	GUI の起動	51
5.3	プロファイルの設定	52
5.3.1	プロファイル ターゲットの選択	53
5.3.2	プロファイル タイプの選択	54
5.3.3	Advanced Options	55

5.3.4	プロファイリングの開始	57
5.4	変換の進行状況	58
5.5	プロファイル データの解析	58
5.5.1	パフォーマンス ホットスポットの概要	59
5.5.2	スレッド同時実行性グラフ	63
5.5.3	関数ホットスポット	64
5.5.4	プロセスと関数	65
5.5.5	ソースとアセンブリ	67
5.5.6	トップダウン コールスタック	68
5.5.7	フレーム グラフ	69
5.5.8	コールグラフ	70
5.5.9	IMIX ビュー	71
5.6	プロファイル データベースのインポート	72
5.7	保存済みのプロファイル セッションの解析	73
5.8	保存されたプロファイル設定の使用	74
5.9	設定	75
5.10	ショートカット キー	77
第 6 章	AMD uProf CLI の使用開始	78
6.1	概要	78
6.2	CPU プロファイルの開始	79
6.2.1	事前定義サンプル設定のリスト	80
6.2.2	プロファイル レポート	81
6.3	消費電力プロファイルの開始	82
6.3.1	システム全体の消費電力プロファイル (ライブ)	82
6.4	収集コマンド	83
6.4.1	オプション	84
6.4.2	Windows 専用オプション	87
6.4.3	Linux 専用オプション	88
6.4.4	例	91
6.5	レポート コマンド	94
6.5.1	オプション	95
6.5.2	Windows 専用オプション	97
6.5.3	Linux 専用オプション	98
6.5.4	例	98

6.6	Translate コマンド	99
6.6.1	オプション	100
6.6.2	Windows 専用オプション	101
6.6.3	Linux 専用オプション	101
6.6.4	例	102
6.7	Timechart コマンド	102
6.7.1	オプション	103
6.7.2	例	103
6.8	Diff コマンド	104
6.8.1	プロファイル比較の適格基準	105
6.8.2	オプション	105
6.8.3	例	107
6.9	Profile コマンド	109
6.9.1	オプション	109
6.9.2	Windows 専用オプション	115
6.9.3	Linux 専用オプション	116
6.9.4	例	119
6.10	Info コマンド	122
6.10.1	オプション	122
6.10.2	例	123
第 7 章	パフォーマンス解析	124
7.1	CPU プロファイリング	124
7.2	時間ベース プロファイリングによる解析	126
7.2.1	プロファイリングの設定と開始	126
7.2.2	プロファイル データの解析	127
7.3	イベント ベース プロファイリングによる解析	128
7.3.1	プロファイリングの設定と開始	128
7.3.2	プロファイル データの解析	129
7.4	命令ベース サンプリングによる解析	129
7.4.1	プロファイリングの設定と開始	129
7.4.2	プロファイル データの解析	130
7.5	コールスタック サンプルによる解析	131
7.5.1	フレーム グラフ	132
7.5.2	コールグラフ	133

7.6	Java アプリケーションのプロファイリング	133
7.6.1	Java アプリケーションの起動	134
7.6.2	Java プロセスからプロファイルへの接続	134
7.6.3	Java ソース ビュー	134
7.6.4	Java コールスタックおよびフレーム グラフ	135
7.7	キャッシュ解析	136
7.7.1	サポートされるメトリクス	137
7.7.2	GUI を使用したキャッシュ解析	137
7.7.3	CLI を使用したキャッシュ解析	138
7.8	カスタム プロファイル	140
7.8.1	プロファイリングの設定と開始	140
7.8.2	プロファイル データの解析	142
7.9	注記	143
7.9.1	信頼性のしきい値	143
7.9.2	問題のしきい値	143
7.10	IBS サンプルの ASCII ダンプ	144
7.11	分岐解析	144
7.12	セッションのエクスポート	146
7.13	制限	146
第 8 章	パフォーマンス解析 (Linux)	148
8.1	スレッド解析	148
8.1.1	CLI を使用したスレッド解析	148
8.1.2	pthread 同期 API	152
8.1.3	libc システム コール ラッパー API	152
8.1.4	Linux のタイムライン解析 GUI	154
8.2	OpenMP の解析	158
8.2.1	GUI を使用した OpenMP アプリケーションのプロファイリング	159
8.2.2	CLI を使用した OpenMP アプリケーションのプロファイリング	160
8.2.3	環境変数	162
8.2.4	制限	163
8.3	MPI のプロファイリング	163
8.3.1	CLI を使用したデータの収集	164
8.3.2	CLI を使用したデータの解析	165
8.3.3	GUI を使用したデータの解析	166

8.3.4	制限	166
8.4	Linux での perf_event_paranoid 値に対するプロファイリング サポート	166
8.5	Linux システム モジュールのプロファイリング	167
8.6	Linux カーネルのプロファイリング	167
8.6.1	カーネル シンボルの解決の有効化	167
8.6.2	カーネル デバッグ シンボル パッケージのダウンロードとインストール	168
8.6.3	デバッグ シンボルを含む Linux カーネルの構築	169
8.6.4	カーネル関数内のホットスポットの解析	169
8.6.5	Linux カーネルのコールスタック サンプリング	169
8.6.6	制約	169
8.7	カーネル ブロック I/O 解析	170
8.7.1	CLI を使用したカーネル ブロック I/O 解析	170
8.8	GPU オフロード解析 (GPU トレース)	172
8.8.1	CLI を使用した GPU オフロード解析	173
8.9	GPU のプロファイリング	175
8.9.1	CLI を使用した GPU プロファイリング	176
8.10	その他の OS トレース イベント	178
8.10.1	CLI を使用したページ フォルトとメモリ割り当てのトレース	178
8.10.2	CLI を使用した関数呼び出しカウンターのトレース	179
8.11	MPI トレース解析	180
8.11.1	CLI を使用した MPI 軽量トレース	181
8.11.2	CLI を使用した MPI フル トレース	183
8.11.3	GUI を使用した MPI フル トレース	188
第 9 章	消費電力プロファイル	193
9.1	概要	193
9.2	メトリクス	193
9.3	GUI でのプロファイルの使用	195
9.3.1	プロファイルの設定	195
9.3.2	プロファイルの解析	196
9.4	CLI を使用したプロファイリング	197
9.4.1	例	198
9.5	AMDPowerProfileAPI ライブラリ	199
9.5.1	API の使用	199
9.6	制限	199

第 10 章	リモート プロファイリング	200
10.1	概要	200
10.2	認可のセットアップ	200
10.3	AMDProfilerService の起動	201
10.4	リモート ターゲットへの接続	202
10.5	制限	204
第 11 章	AMD uProf の仮想化サポート	205
11.1	概要	205
11.2	CPU プロファイリング	207
11.2.1	ゲスト VM からゲスト VM のプロファイリング	207
11.2.2	ホスト システム (KVM ハイパーバイザー) からゲスト VM のプロファイリング	207
11.2.3	ゲスト カーネル モジュールをプロファイリングするためのホスト システムの準備	207
11.2.4	AMD uProf CLI のプロファイリング オプション	207
11.2.5	例	208
11.3	AMDuProfPcm	209
11.4	AMDuProfSys	209
第 12 章	プロファイル制御 API	210
12.1	AMDProfileControl API	210
12.1.1	CPU プロファイル制御 API	210
12.1.2	API の使用	211
12.1.3	インストルメント化されたターゲット アプリケーションのコンパイル	212
12.1.4	インストルメント化されたターゲット アプリケーションのプロファイリング	212
12.1.5	制限	212
第 13 章	参考資料	213
13.1	プロファイリングのためのアプリケーションの準備	213
13.1.1	Windows でのデバッグ情報の生成	213
13.1.2	Linux でのデバッグ情報の生成	214
13.2	CPU プロファイリング	214
13.2.1	ハードウェア ソース	215
13.2.2	プロファイリングの概念	216
13.2.3	プロファイル タイプ	217
13.2.4	事前定義コア PMC イベント	218
13.2.5	IBS の派生イベント	232
13.3	参照用 URL	245

表目次

表 1.	表記規則	17
表 2.	略語	18
表 3.	用語	19
表 1.	ユーザー インターフェイス	2
表 2.	AMDuProfPcm のオプション	10
表 3.	AMD EPYC™ “Zen 2” のパフォーマンス メトリクス	13
表 4.	AMD EPYC™ “Zen 3” のパフォーマンス メトリクス	16
表 5.	AMD EPYC™ “Zen 4” のパフォーマンス メトリクス	20
表 6.	AMDuProfPcm のオプション	26
表 7.	レベル 1 メトリクス	32
表 8.	レベル 2 メトリクス	33
表 9.	AMDuProfSys の一般的なオプション	38
表 10.	AMDuProfSys 収集コマンドのオプション	38
表 11.	AMDuProfSys レポート コマンドのオプション	39
表 12.	サンプル対象データ	43
表 13.	事前定義サンプリング設定	44
表 14.	Assess Performance の設定	46
表 15.	スレッド設定	46
表 16.	Investigate Data Access の設定	46
表 17.	Investigate Branch の設定	47
表 18.	Assess Performance (Extended) の設定	47
表 19.	Investigate Instruction Access の設定	47
表 20.	Investigate CPI の設定	48
表 21.	Instruction Based Sampling の設定	48
表 22.	Summary Overview	60
表 23.	ショートカット キー	77
表 24.	サポートされるコマンド	78
表 25.	AMDuProfCLI の Collect コマンドのオプション	84
表 26.	AMDuProfCLI の Collect コマンド – Windows 専用オプション	87
表 27.	AMDuProfCLI の Collect コマンド – Linux 専用オプション	88
表 28.	AMDuProfCLI の report コマンドのオプション	95
表 29.	AMDuProfCLI の report コマンド – Windows 専用オプション	97

表 30.	AMDuProfCLI の report コマンド – Linux 専用オプション	98
表 31.	AMDuProfCLI Translate コマンドのオプション	100
表 32.	Translate コマンド – Windows 専用オプション	101
表 33.	Translate コマンド – Linux 専用オプション	101
表 34.	AMDuProfCLI Timechart コマンドのオプション	103
表 35.	AMDuProfCLI diff コマンドのオプション	105
表 36.	AMDuProfCLI profile コマンドのオプション	109
表 37.	AMDuProfCLI profile コマンドの Windows オプション	115
表 38.	AMDuProfCLI profile コマンドの Linux オプション	116
表 39.	AMDuProfCLI Info コマンドのオプション	122
表 40.	AMDuProfCLI の Info コマンド – Linux 専用オプション	123
表 41.	IBS オペレーション由来のメトリクス	137
表 42.	並べ替え基準メトリック	139
表 43.	サポートされる CPU イベント	156
表 44.	CPU トレース カテゴリ	156
表 45.	サポート マトリックス	158
表 46.	MPI プロファイリングのサポート マトリックス	164
表 47.	Linux での perf_event_paranoid 値のプロファイリング	166
表 48.	I/O オペレーション	170
表 49.	GPU トレースがサポートされるインターフェイス	172
表 50.	GPU プロファイリングでサポートされるイベント	175
表 51.	GPU プロファイリングでサポートされるメトリクス	176
表 52.	OS トレースでサポートされるイベント	178
表 53.	サポート マトリックス	180
表 54.	軽量トレースでサポートされる MPI API	181
表 55.	MPI API	184
表 56.	Family 17h Model 00h ~ 0Fh (AMD Ryzen™、AMD Ryzen ThreadRipper™、 第 1 世代 AMD EPYC™)	193
表 57.	Family 17h Model 10h ~ 1Fh (AMD Ryzen™、AMD Ryzen™ PRO APU)	194
表 58.	Family 17h Model 70h ~ 7Fh (第 3 世代 AMD Ryzen™)	194
表 59.	Family 17h Model 30h ~ 3Fh (EPYC 7002)	194
表 60.	Family 19h Model 0h ~ 2Fh (EPYC 7003、EPYC 9000)	194
表 61.	AMDProfilerService のオプション	201
表 62.	AMD uProf の仮想化サポート	205
表 63.	AMD uProf CLI Collect コマンドのオプション	208

表 64.	事前定義コア PMC イベント.....	218
表 65.	コア CPU メトリクス.....	230
表 66.	IBS フェッチ イベント.....	232
表 67.	IBS フェッチ メトリクス.....	235
表 68.	IBS オペレーション イベント.....	235
表 69.	AMD “Zen4” および AMD “Zen3” サーバー プラットフォームでの IBS オペレーション メトリクス.....	242

図目次

図 1.	サンプル ルーフライン チャート	32
図 2.	サンプル レポート	33
図 3.	AMD uProf GUI	50
図 4.	AMD uProf の [Welcome] 画面	51
図 5.	[Start Profiling] - [Select Profile Target]	53
図 6.	[Start Profiling] - [Select Profile Configuration]	54
図 7.	[Start Profiling] - [Advanced Options] 1	55
図 8.	[Start Profiling] - [Advanced Options] 2	56
図 9.	プロファイル データの収集	57
図 10.	変換の進行状況	58
図 11.	[Summary] - [Hot Spots]	59
図 12.	[OS Trace]	61
図 13.	[GPU Trace]	61
図 14.	[Summary] - [Thread Concurrency Graph]	63
図 15.	[ANALYZE] - [Function Hotspots]	64
図 16.	[Analyze] - [Metrics]	65
図 17.	[SOURCES] - ソースとアセンブリ	67
図 18.	[Top-down Callstack]	68
図 19.	[ANALYZE] - [Flame Graph]	69
図 20.	[ANALYZE] - [Call Graph]	70
図 21.	IMIX ビュー	71
図 22.	[Import Session] - プロファイル データベースのインポート	72
図 23.	[PROFILE] - [Recent Session(s)]	73
図 24.	[PROFILE] - [Saved Configurations]	74
図 25.	[SETTINGS] - [Preferences]	75
図 26.	[SETTINGS] - [Symbols]	75
図 27.	[SETTINGS] - [Source Data]	76
図 28.	[Profile Data]	76
図 29.	Collect コマンドと Report コマンド	79
図 30.	Linux でサポートされる事前定義設定	80
図 31.	Windows でサポートされる事前定義設定	81
図 32.	timechart --list コマンドの出力	83

図 33.	timechart の実行	83
図 34.	時間ベース プロファイル - 設定	127
図 35.	イベント ベース プロファイル - 設定	128
図 36.	IBS の設定	130
図 37.	[Start Profiling] - [Advanced Options]	131
図 38.	[ANALYZE] - [Flame Graph]	132
図 39.	[ANALYZE] - [Call Graph]	133
図 40.	Java メソッド - ソース ビュー	135
図 41.	Java アプリケーション - フレーム グラフ	136
図 42.	[Cache Analysis]	138
図 43.	キャッシュ解析 - サマリ セクション	139
図 44.	キャッシュ解析 - 詳細レポート	139
図 45.	CPU トレース	141
図 46.	GPU トレース	141
図 47.	[Custom Config] - [Added Categories]	142
図 48.	CPI メトリクス - しきい値ベースのパフォーマンス	143
図 49.	分岐解析サマリ	145
図 50.	トレース レポート	150
図 51.	Linux のタイムライン解析 GUI	155
図 52.	OpenMP トレースの有効化	159
図 53.	[HPC] - [Overview]	159
図 54.	[HPC] - [Parallel Regions]	160
図 55.	OpenMP レポート	161
図 56.	ディスク I/O サマリ テーブル	171
図 57.	解析 - ブロック I/O 統計	172
図 58.	GPU トレース レポート	174
図 59.	GPU プロファイル レポート	177
図 60.	ページフォルトおよびメモリ割り当てのサマリ	179
図 61.	関数カウンターのサマリ	179
図 62.	LWT レポート	182
図 63.	MPI コミュニケーター サマリ テーブル	186
図 64.	MPI ランク サマリ テーブル	186
図 65.	MPI API サマリ テーブル	187
図 66.	MPI コミュニケーション マトリックス	187

図 67.	MPI 集合型 API サマリ テーブル	187
図 68.	プロファイル セッションのインポート	188
図 69.	MPI コミュニケーション マトリックス	188
図 70.	MPI ランク タイムライン	189
図 71.	[MPI P2P API Summary]	190
図 72.	[MPI Collective API Summary]	190
図 73.	システム全体のライブ消費電力プロファイル	195
図 74.	[Timechart] ページ	196
図 75.	--list コマンドの出力	197
図 76.	Timechart の実行	198
図 77.	[Client ID]	200
図 78.	リモート プロファイリングの接続の確立	201
図 79.	IP の選択	202
図 80.	リモート マシンへの接続	202
図 81.	リモート ターゲットのデータ	203
図 82.	[Disconnect] ボタン	203
図 83.	[AMDTClassicMatMul Property Page]	214

改訂履歴

日付	内容	説明
2024 年 1 月	4.2	軽微な編集および更新を実施
2023 年 8 月	4.1	AMD uProf 4.1 機能を追加
2022 年 11 月	4.0	AMD uProf 4.0 機能を追加
2022 年 7 月	3.6	次の内容を追加 • 第 11 章と第 12 章 • セクション 1.2.4、1.2.5、3.4、4.2.1、4.3、5.4.8、6.6、8.10.2、13.1.5 第 9 章で、古い APU ファミリ向けのサポート対象カウンター カテゴリを削除 一般的な編集とリリース関連の更新の追加
2022 年 1 月	3.5	AMD uProf 3.5 機能を追加
2021 年 4 月	初期	AMD uProf 3.4 機能を文書化

このユーザー ガイドについて

このドキュメントでは、AMD uProf を使用して、AMD プロセッサ ベースの、Windows[®]、Linux[®]、FreeBSD[®] オペレーティング システム上のアプリケーションが動作する際の、CPU、GPU、消費電力を解析する方法を説明します。

このドキュメントの最新バージョンについては、AMD uProf のウェブサイト (<https://www.amd.com/ja/developer/uprof.html>) を参照してください。

対象者

このドキュメントの対象者は、アプリケーションのパフォーマンス改善を必要とするソフトウェア開発者およびパフォーマンス チューニング エキスパートです。対象者が、CPU アーキテクチャ、スレッド、プロセス、ロード モジュールの概念を理解しており、パフォーマンス解析の概念に精通していることを前提としています。

表記規則

このドキュメントで使用されている表記規則は次のとおりです。

表 1. 表記規則

表記規則	説明
GUI 要素	メニュー バーやボタンなどのグラフィカル ユーザー インターフェイス要素
>	メニュー内のメニュー項目
[]	構文内でオプションとなっている項目
...	先行する要素を繰り返し可能
	「または」を意味する。例: 2 つのオプションを同時に指定できない
ファイル名	ファイル、パス、またはソース コード スニペットの名前
コマンド	コマンド名またはコマンド フレーズ
ハイパーリンク	外部ウェブサイトへのリンク

略語

このドキュメントで使用されている略語は次のとおりです。

表 2. 略語

略語	説明
APERF	実パフォーマンス周波数クロック カウンター
ASLR	アドレス空間配置のランダム化
CCD	1 つ以上の CCX と、IOD に接続する GMI2 ファブリック ポートを格納できる コア コンプレックス ダイ
CLI	コマンド ライン インターフェイス
CPI	命令あたりのサイクル数
CSV	カンマ区切り 値形式
DC	データ キャッシュ
DIMM	デュアル インライン メモリ モジュール
DRAM	ダイナミック ランダム アクセス メモリ
DTLB	データ変換ルックアサイド バッファ
EBP	イベント ベース プロファイリング、コア PMC イベントを使用
GUI	グラフィカル ユーザー インターフェイス
IBS	命令ベースのサンプリング
IC	命令キャッシュ
IOD	IO ダイ
IPC	サイクルあたりの命令数
ITLB	命令変換ルックアサイド バッファ
MPERF	最大パフォーマンス周波数クロック カウンター
MSR	モデル固有レジスタ
NB	ノースブリッジ
OS	オペレーティング システム
P0Freq	P0 ステート周波数
PMC	パフォーマンス モニター カウンター
PTI	1000 命令あたり
RAPL	移動平均消費電力制限
SMU	システム管理ユニット
TBP	時間ベース プロファイリング
TSC	タイムスタンプ カウンター
UMC	ユニファイド メモリ コントローラー 最大 8 の UMC、それぞれがソケットあたり 1 つの DRAM チャンネルをサポート、 各チャンネルに最大 2 の DIMM を構成可能

用語

このドキュメントで使用されている用語は次のとおりです。

表 3. 用語

用語	説明
AMD uProf	製品名。
AMDuProfGUI	グラフィカル ユーザー インターフェイス ツールの名前。
AMDuProfCLI	コマンド ライン インターフェイス ツールの名前。
AMDuProfPcm	システム解析用コマンド ライン インターフェイス ツールの名前。
AMDuProfSys	Python ベースのシステム解析用コマンド ライン インターフェイス ツールの名前。
クライアント	ホスト システム上で動作する AMD uProf または AMDuProfCLI のインスタンス。
コア	論理コア数。SMT 構成に応じて、1 つのコアに 1 つまたは 2 つの CPU を搭載可能。
コア コンプレックス (CCX)	1 つ以上のコアとキャッシュ システムによる構成。
CPU	オペレーティング システムによって認識される論理 CPU 数。
ホスト システム	AMD uProf クライアント プロセスが実行されるシステム。
L1D、L1I キャッシュ	CPU 排他データおよび命令キャッシュ。
L2 キャッシュ	コア内すべての CPU により共有されるキャッシュ。
L3 キャッシュ	CCX 内のすべての CPU により共有されるキャッシュ。
ノード	論理 NUMA ノード。
パフォーマンスプロファイリング (または) CPU プロファイリング	パフォーマンス ボトルネックの特定と解析。パフォーマンス プロファイリングと CPU プロファイリングは同じことを意味する。
ソケット	論理ソケット数。1 つのソケットに複数のノードを構成可能。
システム解析	AMDuProfPcm または AMDuProfSys ツールを参照。
ターゲット システム	プロファイル データが収集されるシステム。

パート 1: はじめに

第 1 章 はじめに

1.1 概要

AMD uProf は、Windows および Linux オペレーティング システム上で動作するアプリケーション向けのパフォーマンス解析ツールです。開発者がアプリケーションのランタイム パフォーマンスを理解し、改善するのに役立ちます。

AMD uProf は次の機能を提供します。

- パフォーマンス解析 (CPU プロファイル)

アプリケーションのランタイム パフォーマンスのボトルネックを特定します。

- システム解析

IPC とメモリ帯域幅などのシステム パフォーマンス メトリクスを監視します。

- ライブ消費電力プロファイリング

システムの熱特性と消費電力特性を監視します。

AMD uProf には次のユーザー インターフェイスがあります。

表 1. ユーザー インターフェイス

実行ファイル	説明	サポートされる OS
AMDuProf	CPU および消費電力プロファイリングを実行するための GUI	Windows、Linux
AMDuProfCLI	CPU および消費電力プロファイリングを実行するための CLI	Windows、Linux、FreeBSD
AMDuProfPcm	システム解析を実行するための CLI	Windows、Linux、FreeBSD
AMDPerf/ AMDuProfSys.py	システム解析のための Python スクリプト	Windows、Linux

AMD uProf は、次の目的に対して効果的に使用できます。

- 1 つ以上のプロセス/アプリケーションのパフォーマンスを解析する。
- ソース コード内のパフォーマンス ボトルネックを突き止める。
- パフォーマンスと電力効率を引き上げるためにソース コードを最適化する方法を特定する。
- カーネル、ドライバ、システム モジュールの動作を調べる。
- システム レベルの熱特性および消費電力特性を観察する。
- IPC とメモリ帯域幅などのシステム メトリクスを観察する。

1.2 仕様

AMD uProf は次の仕様をサポートしています。サポートされるプロセッサとオペレーティング システムの詳細なリストについては、次のウェブサイトからアクセスできる『AMD uProf リリース ノート』を参照してください。

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

1.2.1 プロセッサ

- AMD “Zen” ベースの CPU および APU プロセッサ
- AMD Instinct™ MI100 および MI200 アクセラレータ (GPU カーネルのプロファイリングとトレース用)
- Intel® プロセッサ (時間ベース プロファイリングのみ)

1.2.2 オペレーティング システム

AMD uProf は、次のオペレーティング システムの 64 ビット版をサポートしています。

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

AMD EPYC™ プロセッサの OS サポートについては、AMD のウェブサイト (<https://www.amd.com/ja/processors/epyc-minimum-operating-system>) を参照してください。

1.2.3 コンパイラとアプリケーション環境

AMD uProf は次のアプリケーション環境をサポートしています。

- 言語
 - ネイティブ言語: C、C++、Fortran、アセンブリ
 - 非ネイティブ言語: Java、C#
- 次のコンパイラを使用したプログラム
 - Microsoft コンパイラ、GNU コンパイラ、LLVM
 - AMD Optimizing C/C++ and Fortran Compilers (AOCC)
 - Intel コンパイラ (ICC)

- 並列処理
 - OpenMP
 - MPI
- デバッグ情報の形式: PDB、COFF、DWARF、STABS
- 最適化またはデバッグ情報あり/なしでコンパイルされたアプリケーション
- シングルプロセス、マルチプロセス、シングルスレッド、マルチスレッドのアプリケーション
- 動的にリンク/ロードされるライブラリ
- Windows 上の POSIX 開発環境
 - Cygwin
 - MinGW

1.2.4 仮想化のサポート

AMD uProf は仮想化環境で使用できます。ただし、ハードウェア パフォーマンス カウンターへのアクセスに関する制限がある場合があります。詳細は、[205 ページの「AMD uProf の仮想化サポート」](#)を参照してください。サポートされる仮想化環境は、次のとおりです。

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

1.2.5 コンテナのサポート

AMD uProf CPUProfiler を使用して、Docker コンテナ環境内で動作するアプリケーションを解析できます。これは、Linux プラットフォームのみでサポートされています。アプリケーション解析には、次のいずれかの方法を使用します。

- AMD uProf を Docker コンテナ内で実行して、アプリケーションを解析します。プロファイリングを有効にするには、CAP_SYS_ADMIN 権限 (`docker run --cap-add=CAP_SYS_ADMIN`) が必要です。このモードでは、CLI ベースと GUI ベースの両方のプロファイリングおよび解析がサポートされています。
- AMD uProf CLI を Docker コンテナ外部で実行し、コンテナ内で動作するターゲット アプリケーションをプロファイリングし、解析します。
 - 収集中に `--pid` オプションを使用して、コンテナ化されたプロセスに uProf CLI を接続します。または、システム全体のデータを収集して、レポート生成中に PID でフィルターします。
 - レポート生成中に、コンテナ内で実行されているプロファイリング対象アプリケーションのバイナリとソース コードへのパス (`--bin-path` および `--src-path`) を指定します。AMD uProf GUI では、このモードでのプロファイリングと解析はサポートされていません。

1.3 AMD uProf のインストール

AMD ポータル (<https://www.amd.com/ja/developer/uprof.html>) から、サポートされるオペレーティング システムの AMD uProf インストーラー パッケージの最新バージョンをダウンロードします。インストール方法は次のとおりです。

1.3.1 Windows

64 ビット Windows のインストーラー バイナリ *AMDuProf-x.y.z.exe* を実行します。

インストールが完了すると、実行ファイル、ライブラリ、その他の必要なファイルが、*C:\Program Files\AMD\AMDuProf* フォルダーにインストールされます。

1.3.2 Linux

1.3.2.1 tar ファイルを使用したインストール

次のコマンドを使用し、tar.bz2 バイナリ ファイルを解凍して AMD uProf をインストールします。

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

注記: Power Profiler Linux Driver を手動でインストールする必要があります。

1.3.2.2 RPM パッケージを使用したインストール (RHEL)

次の rpm または yum コマンドを使用して、AMD uProf RPM パッケージをインストールします。

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

インストールが完了すると、実行ファイル、ライブラリ、その他の必要なファイルが、*/opt/AMDuProf_X.Y-ZZZ/* ディレクトリにインストールされます。

1.3.2.3 Debian パッケージを使用したインストール (Ubuntu)

次の dpkg コマンドを使用して、AMD uProf Debian パッケージをインストールします。

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

インストールが完了すると、実行ファイル、ライブラリ、その他の必要なファイルが、*/opt/AMDuProf_X.Y-ZZZ/* ディレクトリにインストールされます。

1.3.2.4 Linux への Power Profiler Driver のインストール

RPM および Debian インストーラー パッケージを使用して AMD uProf をインストールする場合、Power Profiler Linux Driver ビルドが生成されて、自動的にインストールされます。ただし、AMD uProf tar.bz2 アーカイブをダウンロードした場合は、Power Profiler Linux Driver を手動でインストールする必要があります。

Power Profiler Driver をインストールするには、GCC および MAKE ソフトウェア パッケージが前提条件として必要です。これらのパッケージをインストールしていない場合は、次のコマンドでインストールできます。

RHEL および CentOS ディストリビューション:

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

Debian/Ubuntu ディストリビューション:

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

次のコマンドを実行します。

```
$ tar -xf AMDuProf_Linux_x64_x.y.z.tar.bz2
$ cd AMDuProf_Linux_x64_x.y.z/bin
$ sudo ./AMDPowerProfilerDriver.sh install
```

インストーラーによって、`/usr/src/AMDPowerProfiler-<バージョン>` ディレクトリ内に Power Profiler Driver のソース ツリーが作成されます。モジュールのコンパイルに必要なすべてのソース ファイルがこのディレクトリに含まれており、MIT ライセンスが適用されます。

このドライバーをアンインストールするには、次のコマンドを実行します。

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

1.3.2.5 Linux 消費電力プロファイリングドライバーの DKMS のサポート

Linux マシンでは、Dynamic Kernel Module Support (DKMS) フレームワークのサポート付きで Power Profiler Driver をインストールすることもできます。DKMS フレームワークでは、既存のカーネルに変更が加わると、Power Profiler Driver モジュールが自動的にアップグレードされます。このため、消費電力プロファイリングドライバー モジュールを手動でアップグレードする必要がなくなります。DKMS パッケージは、前のセクションに記載したインストール手順を実行する前にターゲット マシンにインストールしておく必要があります。DKMS パッケージがターゲット マシンにインストールされている場合、`AMDPowerProfilerDriver.sh` インストーラーによって DKMS 関連の設定が自動的に処理されます。

Ubuntu ディストリビューションの例

```
$ 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
```

カーネル バージョンを頻繁にアップグレードする場合は、インストール時に DKMS を使用することを推奨します。

1.3.2.6 ROCm のインストール

『ROCm インストール ガイド』(https://docs.amd.com/bundle/ROCm-Installation-Guide-v5.5/page/Introduction_to_ROCm_Installation_Guide_for_Linux.html) に記載されている手順を完了して、AMD ROCm™ v5.5 をホスト システムにインストールします。

ROCm 5.5 をインストールした後で、`/opt/rocm/` のシンボリック リンクが `/opt/rocm-5.5.0/` を参照していることを確認します。

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

AMD ROCm v5.5 のインストールは、GPU のトレースとプロファイリングに必要です。

1.3.2.7 BCC のインストールと eBPF

BCC をインストールするには、BCC ウェブサイト (<https://github.com/iovisor/bcc/blob/master/INSTALL.md>) に記載された手順を完了します。

BCC をインストールした後で、次のコマンドを実行して BCC のインストールを確認します。

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

RPM/DEB インストーラーを使用して AMD uProf をインストールする場合、このインストーラーによってスクリプトが実行され、ホストでの BCC のインストールと eBPF (Extended Berkeley Packet Filter) のサポートに関する情報が提供されます。

1.3.3 FreeBSD

`tar.bz2` バイナリ ファイルを解凍し、AMD uProf をインストールします。

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

1.4 サンプル プログラム

ツールの使用法を示すいくつかのサンプル プログラムが、製品と共にインストールされます。

- Windows

サンプルの行列乗算アプリケーション

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

- Linux

- サンプルの行列乗算プログラムと `makefile`

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

- OpenMP サンプル プログラムおよびそのバリエーションと `makefile`

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

- FreeBSD

サンプルの行列乗算プログラムと `makefile`

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

1.5 サポート

サポート オプション、最新の資料、ダウンロードについては、AMD ポータル (<https://www.amd.com/ja/developer/uprof.html>) を参照してください。

パート 2: システム解析

第 2 章 AMDuProfPcm の使用開始

2.1 概要

システム解析ユーティリティ AMDuProfPcm は、Family 17h および 19h の AMD EPYC™ 7001、AMD EPYC™ 7002、AMD EPYC™ 7003、AMD EPYC™ 9000 プロセッサで、基本的なパフォーマンス監視メトリクスをモニタリングするのに役立ちます。このユーティリティは、CPU コア、L3、DF のパフォーマンス イベント カウント値を定期的に収集し、各種メトリクスをレポートします。Windows、Linux、FreeBSD 上でサポートされています。

2.1.1 使用要件

2.1.1.1 Linux

- AMDuProfPcm は、`--msr` を指定したデータ収集時のみ、MSR ドライバーに加えて、ルート権限または `dev/cpu/*/msr` デバイスの読み出し書き込み権限を必要とします。
- NMI ウォッチドッグを無効にする必要があります (`echo 0 > /proc/sys/kernel/nmi_watchdog`)。
- `/proc/sys/kernel/perf_event_paranoid` を「-1」に設定します。
- 次のコマンドを使用して、`msr` ドライバーをロードします。

```
$ modprobe msr
```
- ループラインプロット スクリプト (`AMDuProfModelling.py`) には、python 3.x と python モジュール「matplotlib」が必要です。

2.1.1.2 FreeBSD

AMDuProfPcm は `cpuctl` モジュールを使用し、ルート権限または `/dev/cpuctl*` デバイスの読み出し書き込み権限のいずれかを必要とします。

構造:

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

<PROGRAM> — プロファイリングの対象となる起動アプリケーションを指定します。

<ARGS> — 起動アプリケーションに渡す引数のリストを指定します。

一般的な使用法:

```
$ AMDuProfPcm -h
# AMDuProfPcm -m ipc -c core=0 -d 10 -o /tmp/pmcdata.txt
# AMDuProfPcm -m memory -a -d 10 -o /tmp/memdata.txt -- /tmp/myapp.exe
```

2.2 オプション

次の表にオプションの一覧を示します。

表 2. AMDuProfPcm のオプション

オプション	説明
-h	このヘルプ情報をコンソール/ターミナルに表示します。
-m <metric,...>	レポート対象となるメトリクス。デフォルト メトリクス グループは「ipc」です。 サポートされるメトリクス グループと対応するメトリクスは、プラットフォーム、OS、ハイパーバイザーによって異なります。 サポートされるメトリクスのリストを表示するには、AMDuProfpcm -h を実行します。 サポートされるメトリクス グループは、次のとおりです。 • ipc – CEF、使用率、CPI、IPC などのメトリクスをレポート • fp – GFLOPS をレポート • l1 – L1 キャッシュ関連のメトリクス (DC アクセスと IC フェッチ ミス率) • l2 – L2D および L2I キャッシュ関連のアクセス/ヒット/ミス メトリクス • l3 – L3 アクセス、L3 ミス、平均ミス レイテンシなどの L3 キャッシュ メトリクス • dc – ソース別 DC リフィルなどの高度なキャッシング メトリクス (AMD “Zen3” および AMD “Zen4” プロセッサのみでサポート) • memory – 全チャネルのメモリ読み出しおよび書き込み帯域幅 (概算値、GB/s) • pcie – PCIe 帯域幅 (GB/s、AMD “Zen2” および AMD “Zen4” プロセッサのみでサポート) • xgmi – 全リモート リンクの xGMI 送信データバイト (概算値、GB/s) • dma – DMA 帯域幅 (GB/s、AMD “Zen4” プロセッサのみでサポート) • swpfdc – 各種ノードからのソフトウェアプリフェッチ データ キャッシュと CCX (AMD “Zen3” および AMD “Zen4” プロセッサのみでサポート) • hwpfdc – 各種ノードからのハードウェアプリフェッチ データ キャッシュと CCX (AMD “Zen3” および AMD “Zen4” プロセッサのみでサポート) • pipeline_util – CPU パイプライン内のボトルネックを可視化するためのトップダウン メトリクス (AMD “Zen4” プロセッサのみでサポート)

表 2. AMDuProfPcm のオプション (続き)

オプション	説明
-c <core ccx l3 ccd package>=<n>	収集対象として core ccx ccd package のいずれかを指定します。デフォルト値は「core=0」です。 「ccx」または「l3」が指定された場合: - この ccx のすべてのコアからコア イベントが収集されます。 - この ccx の最初のコアから l3 および df イベントが収集されます。 「ccd」が指定された場合: - このダイのすべてのコアからコア イベントが収集されます。 - このダイのすべての ccx の最初のコアから l3 イベントが収集されます。 - このダイの最初のコアから df イベントが収集されます。 「package」が指定された場合: - このパッケージのすべてのコアからコア イベントが収集されます。 - このパッケージのすべての ccx の最初のコアから l3 イベントが収集されます。 - このパッケージのすべてのダイの最初のコアから df イベントが収集されます。
-a	すべてのコアから収集されます。 注記: オプション -c と -a は同時に使用できません。
-C	プロファイリング時間の最後に累積データを出力します。そうでない場合は、すべてのサンプルが時系列データとしてレポートされます。
-A <system,package,ccd,ccx,core>	各種コンポーネント レベルで集計されたメトリクスを出力します。次の粒度がサポートされます。 system – システム内すべてのコアから収集されたサンプルが集計されます。 package – システム内にあるすべてのパッケージに対して、パッケージ内すべてのコアから収集されたサンプルが集計され、レポートされます。マルチパッケージシステムに適用されます。 ccd – すべての CCD に対して、CCD 内すべてのコアから収集されたサンプルが集計され、レポートされます。 ccx – すべての CCX に対して、CCX 内すべてのコアから収集されたサンプルが集計され、レポートされます。 core – サンプル収集対象であるすべてのコアから取得されたサンプルが、集計なしでレポートされます。 注記: - すべてのコアからサンプルを収集するには、このオプションと一緒にオプション -a を使用する必要があります。 - コンポーネントのカンマ区切りリストを指定できます。

表 2. AMDuProfPcm のオプション (続き)

オプション	説明
-i <config file>	監視対象の Core[L3]DF カウンターを指定したユーザー定義の XML 設定ファイル。 フォーマットについては、<インストールディレクトリ>/bin/Data/Config/ ディレクトリ内のサンプルファイルを参照してください。 注記: 1. オプション -i と -m は同時に使用できません。 2. オプション -i を使用すると、ユーザー定義設定ファイル内に指定されたすべてのイベントが収集されます。
-d <seconds>	実行するプロファイリングの長さ。
-t < multiplex interval in ms>	pmc カウント値が読み出される間隔。最小値は 16 ms です。
-o <output file>	出力ファイル名。ファイルは CSV 形式です。
-D <dump file>	すべての監視対象イベントのイベント カウント ダンプを含む出力ファイル。ファイルは CSV 形式です。
-p <n>	レポートされるメトリクスの精度を設定します。デフォルト値は 2 です。
-q	出力レポートの CPU トポロジ セクションを非表示にします。
-r	MSR を強制リセットします。
-k	パッケージレベル カウンターに接頭辞「pkg」を付けます。
-s	時系列レポートにタイムスタンプを表示します。
-l	サポートされる生の PMC イベント リストを出力します。
-z <pmc-event>	イベントの名前、説明、使用可能なユニット マスクを出力します。
-x <core-id,...>	起動されたアプリケーションのコアのアフィニティ。コア ID のカンマ区切りリスト。 注記: これは、Linux のみでサポートされています。
-w <dir>	作業ディレクトリを指定します。デフォルトは、起動されたアプリケーションのパスになります。
-v	バージョンを出力します。
-X	perf サブシステムを使用して、ルート権限なしでデータを収集します。 注記: これは、Linux のみでサポートされています。
-P <process ID>	監視するターゲットのプロセス ID を指定します。 注記: これは、Linux でオプション -X を使用した場合のみサポートされています。
-f <util:<n>>	使用率に基づいてルーフライン データをフィルターします。たとえば、「-f util:90」は、使用率が 90% 未満のすべてのデータポイントをフィルターします。 注記: これが適用されるのは、roofline コマンドと一緒に使用された場合のみです。

AMD EPYC™ “Zen 2” コア アーキテクチャ プロセッサのパフォーマンス メトリクスは、次のとおりです。

表 3. AMD EPYC™ “Zen 2” のパフォーマンス メトリクス

メトリクス グループ	メトリクス	説明
ipc	Utilization (%)	コアが実行状態で、アイドル時間ではなかった時間の割合。
	Eff Freq	サンプリング時間全体にわたる、一時停止サイクルなしのコア有効周波数 (CEF) を、GHz でレポートします。このメトリクスのベースとなる式は次のとおりです。CEF = (APERF / TSC) * P0Freq。コアが C6 ステートのとき、APERF は実際のコアサイクル数に比例してインクリメントします。
	IPC	サイクルあたりの実行命令数 (IPC) は、1 CPU サイクルあたりにリタイアした平均の命令数です。これは、コア PMC イベントである PMCx0C0 [Retired Instructions] と PMCx076 [CPU Clocks not Halted] を使用して測定されます。これらの PMC イベントは、OS とユーザー モードの両方でカウントされます。
	CPI	命令あたりのサイクル数 (CPI) は、IPC メトリクスの逆数です。これは、キャッシュ ミス、分岐予測ミス、メモリ レイテンシ、その他のボトルネックにより、アプリケーションの実行にどう影響が及ぶかを示す基本的なパフォーマンス メトリクスの 1 つです。CPI 値が低いほど、パフォーマンスは良くなります。
	Branch Mis-prediction Ratio	予測ミスした分岐とリタイアした分岐命令の割合。
fp	Retired SSE/AVX Flops (GFLOPs)	リタイアした SSE/AVX FLOPS の数。
	Mixed SSE/AVX Stalls	SSE/AVX の混合ストール。 このメトリクスの単位は、1,000 命令あたり (PTI) です。
l1	IC(32B) Fetch Miss Ratio	命令キャッシュのフェッチ ミス率。
	DC Access	すべてのデータ キャッシュ (DC) アクセス。このメトリクスの単位は PTI です。

表 3. AMD EPYC™ “Zen 2” のパフォーマンス メトリクス (続き)

メトリクス グループ	メトリクス	説明
l2	L2 Access	すべての L2 キャッシュ アクセス。このメトリクスの単位は PTI です。
	L2 Access from IC Miss	IC ミスからの L2 キャッシュ アクセス。このメトリクスの単位は PTI です。
	L2 Access from DC Miss	DC ミスからの L2 キャッシュ アクセス。このメトリクスの単位は PTI です。
	L2 Access from HWPf	L2 ハードウェア プリフェッチからの L2 キャッシュ アクセス。このメトリクスの単位は PTI です。
	L2 Miss	すべての L2 キャッシュ ミス。このメトリクスの単位は PTI です。
	L2 Miss from IC Miss	IC ミスからの L2 キャッシュ ミス。このメトリクスの単位は PTI です。
	L2 Miss from DC Miss	DC ミスからの L2 キャッシュ ミス。このメトリクスの単位は PTI です。
	L2 Miss from HWPf	L2 ハードウェア プリフェッチからの L2 キャッシュ ミス。このメトリクスの単位は PTI です。
	L2 Hit	すべての L2 キャッシュ ヒット。このメトリクスの単位は PTI です。
	L2 Hit from IC Miss	IC ミスからの L2 キャッシュ ヒット。このメトリクスの単位は PTI です。
	L2 Hit from DC Miss	DC ミスからの L2 キャッシュ ヒット。このメトリクスの単位は PTI です。
	L2 Hit from HWPf	L2 ハードウェア プリフェッチからの L2 キャッシュ ヒット。このメトリクスの単位は PTI です。

表 3. AMD EPYC™ “Zen 2” のパフォーマンス メトリクス (続き)

メトリクス グループ	メトリクス	説明
tlb	L1 ITLB Miss	L1 命令変換ルックアサイド バッファ (ITLB) はミスとなったが、L2-ITLB とページ テーブル ウォークによる ITLB リロードではヒットした命令フェッチ。テーブル ウォーク要求の対象は、L1-ITLB ミスと L2-ITLB ミスです。このメトリクスの単位は PTI です。
	L2 ITLB Miss	L1-ITLB ミスと L2-ITLB ミスに起因したページ テーブル ウォークによる ITLB リロードの数。このメトリクスの単位は PTI です。
	L1 DTLB Miss	ロード/ストアのマイクロオペレーションによる L1 データ変換ルックアサイド バッファ (DTLB) ミスの数。このイベントでは、L2-DTLB ヒットと L2-DTLB ミスの両方がカウントされます。このメトリクスの単位は PTI です。
	L2 DTLB Miss	ロード/ストアのマイクロオペレーションによる L2 データ変換ルックアサイド バッファ (DTLB) ミスの数。このメトリクスの単位は PTI です。
l3	L3 Access	L3 キャッシュ アクセス。このメトリクスの単位は PTI です。
	L3 Miss	L3 キャッシュ ミス。このメトリクスの単位は PTI です。
	L3 Miss (%)	L3 キャッシュ ミスの割合。このメトリクスの単位は PTI です。
	Ave L3 Miss Latency	コア サイクルでの平均 L3 ミス レイテンシ。
Memory	Mem Ch-A RdBw (GB/s) Mem Ch-A WrBw (GB/s) ...	全チャネルのメモリ読み出しおよび書き込み帯域幅 (GB/s)。
xgmi	xGMI0 BW (GB/s) xGMI1 BW (GB/s) xGMI2 BW (GB/s) xGMI3 BW (GB/s)	全リモート リンクの xGMI 送信データ バイト (概算値、GB/s)。
pcie	PCIe0 (GB/s) PCIe1 (GB/s) PCIe2 (GB/s) PCIe3 (GB/s)	PCIe 帯域幅 (概算値、GB/s)。

AMD EPYC™「Zen 3」コア アーキテクチャ プロセッサのパフォーマンス メトリクスは、次のとおりです。

表 4. AMD EPYC™「Zen 3」のパフォーマンス メトリクス

メトリクス グループ	メトリクス	説明
ipc	Utilization (%)	コアが実行状態で、アイドル時間ではなかった時間の割合。
	Eff Freq	サンプリング時間全体にわたる、一時停止サイクルなしのコア有効周波数 (CEF) を、GHz でレポートします。このメトリクスのベースとなる式は次のとおりです。CEF = (APERF / TSC) * P0Freq。コアが C6 ステートするとき、APERF は実際のコアサイクル数に比例してインクリメントします。
	IPC	サイクルあたりの実行命令数 (IPC) は、1 CPU サイクルあたりにリタイアした平均の命令数です。これは、コア PMC イベントである PMCx0C0 [Retired Instructions] と PMCx076 [CPU Clocks not Halted] を使用して測定されます。これらの PMC イベントは、OS とユーザー モードの両方でカウントされます。
	CPI	命令あたりのサイクル数 (CPI) は、IPC メトリクスの逆数です。これは、キャッシュ ミス、分岐予測ミス、メモリ レイテンシ、その他のボトルネックにより、アプリケーションの実行にどう影響が及ぶかを示す基本的なパフォーマンス メトリクスの 1 つです。CPI 値が低いほど、パフォーマンスは良くなります。
	Branch Mis-prediction Ratio	予測ミスした分岐とリタイアした分岐命令の割合。
fp	Retired SSE/AVX Flops (GFLOPs)	リタイアした SSE/AVX FLOPS の数。
	Mixed SSE/AVX Stalls	SSE/AVX の混合ストール。 このメトリクスの単位は、1,000 命令あたり (PTI) です。
l1	IC (32B) Fetch Miss Ratio	命令キャッシュのフェッチ ミス率。
	Op Cache (64B) Fetch Miss Ratio	オペレーション キャッシュのフェッチ ミス率。
	IC Access	すべての命令キャッシュ アクセス。このメトリクスの単位は PTI です。
	IC Miss	命令キャッシュ ミス。このメトリクスの単位は PTI です。
	DC Access	すべての DC アクセス。このメトリクスの単位は PTI です。

表 4. AMD EPYC™ 「Zen 3」 のパフォーマンス メトリクス (続き)

メトリクス グループ	メトリクス	説明
l2	L2 Access	すべての L2 キャッシュ アクセス。このメトリクスの単位は PTI です。
	L2 Access from IC Miss	IC ミスからの L2 キャッシュ アクセス。このメトリクスの単位は PTI です。
	L2 Access from DC Miss	DC ミスからの L2 キャッシュ アクセス。このメトリクスの単位は PTI です。
	L2 Access from HWPF	L2 ハードウェア プリフェッチからの L2 キャッシュ アクセス。このメトリクスの単位は PTI です。
	L2 Miss	すべての L2 キャッシュ ミス。このメトリクスの単位は PTI です。
	L2 Miss from IC Miss	IC ミスからの L2 キャッシュ ミス。このメトリクスの単位は PTI です。
	L2 Miss from DC Miss	DC ミスからの L2 キャッシュ ミス。このメトリクスの単位は PTI です。
	L2 Miss from HWPF	L2 ハードウェア プリフェッチからの L2 キャッシュ ミス。このメトリクスの単位は PTI です。
	L2 Hit	すべての L2 キャッシュ ヒット。このメトリクスの単位は PTI です。
	L2 Hit from IC Miss	IC ミスからの L2 キャッシュ ヒット。このメトリクスの単位は PTI です。
	L2 Hit from DC Miss	DC ミスからの L2 キャッシュ ヒット。このメトリクスの単位は PTI です。
	L2 Hit from HWPF	L2 ハードウェア プリフェッチからの L2 キャッシュ ヒット。このメトリクスの単位は PTI です。

表 4. AMD EPYC™ 「Zen 3」 のパフォーマンス メトリクス (続き)

メトリクス グループ	メトリクス	説明
tlb	L1 ITLB Miss	L1 命令変換ルックアサイド バッファ (ITLB) はミスとなったが、L2-ITLB とページ テーブル ウォークによる ITLB リロードではヒットした命令フェッチ。テーブル ウォーク要求の対象は、L1-ITLB ミスと L2-ITLB ミスです。このメトリクスの単位は PTI です。
	L2 ITLB Miss	L1-ITLB ミスと L2-ITLB ミスに起因したページ テーブル ウォークによる ITLB リロードの数。このメトリクスの単位は PTI です。
	L1 DTLB Miss	ロード/ストアのマイクロオペレーションによる L1 データ変換ルックアサイド バッファ (DTLB) ミスの数。このイベントでは、L2-DTLB ヒットと L2-DTLB ミスの両方がカウントされます。このメトリクスの単位は PTI です。
	L2 DTLB Miss	ロード/ストアのマイクロオペレーションによる L2 データ変換ルックアサイド バッファ (DTLB) ミスの数。このメトリクスの単位は PTI です。
	All TLBs Flushed	フラッシュされたすべての TLB。このメトリクスの単位は PTI です。
dc	DC Fills from Same CCX	ローカル L2 キャッシュから、同じ CCX 内のコアあるいは異なる L2 キャッシュ、またはその CCX に属する L3 キャッシュへの DC フィル数。このメトリクスの単位は PTI です。
	DC Fills from different CCX in same node	同じパッケージ (ノード) に含まれる異なる CCX のキャッシュからの DC フィル数。このメトリクスの単位は PTI です。
	DC Fills from Local Memory	同じパッケージ (ノード) 内で接続された DRAM または IO からの DC フィル数。このメトリクスの単位は PTI です。
	DC Fills from Remote CCX Cache	異なるパッケージ (ノード) に含まれる CCX のキャッシュからの DC フィル数。このメトリクスの単位は PTI です。
	DC Fills from Remote Memory	異なるパッケージ (ノード) 内で接続された DRAM または IO からの DC フィル数。このメトリクスの単位は PTI です。
	All DC Fills	すべてのデータ ソースからの DC フィルの合計数。このメトリクスの単位は PTI です。
l3	L3 Access	L3 キャッシュ アクセス。このメトリクスの単位は PTI です。
	L3 Miss	L3 キャッシュ ミス。このメトリクスの単位は PTI です。
	L3 Miss (%)	L3 キャッシュ ミスの割合。このメトリクスの単位は PTI です。
	Ave L3 Miss Latency	コア サイクルでの平均 L3 ミス レイテンシ。
Memory	Mem Ch-A RdBw (GB/s) Mem Ch-A WrBw (GB/s) ...	全チャネルのメモリ読み出しおよび書き込み帯域幅 (GB/s)。

表 4. AMD EPYC™ 「Zen 3」 のパフォーマンス メトリクス (続き)

メトリクス グループ	メトリクス	説明
xgmi	xGMI0 BW (GB/s) xGMI1 BW (GB/s) xGMI2 BW (GB/s) xGMI3 BW (GB/s)	全リモート リンクの xGMI 送信データ バイト (概算値、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)	各種ノードおよび CCX からのソフトウェア プリフェッチ データ キャッシュ。
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)	各種ノードおよび CCX からのハードウェア プリフェッチ データ キャッシュ。

AMD EPYC™ “Zen 4” コア アーキテクチャ プロセッサのパフォーマンス メトリクスは、次のとおりです。

表 5. AMD EPYC™ “Zen 4” のパフォーマンス メトリクス

メトリクス グループ	メトリクス	説明
ipc	Utilization (%)	コアが実行状態で、アイドル時間ではなかった時間の割合。
	Eff Freq	サンプリング時間全体にわたる、一時停止サイクルなしのコア有効周波数 (CEF) を、GHz でレポートします。このメトリクスのベースとなる式は次のとおりです。CEF = (APERF / TSC) * P0Freq。コアが C6 ステートするとき、APERF は実際のコアサイクル数に比例してインクリメントします。
	IPC	サイクルあたりの実行命令数 (IPC) は、1 CPU サイクルあたりにリタイアした平均の命令数です。これは、コア PMC イベントである PMCx0C0 [Retired Instructions] と PMCx076 [CPU Clocks not Halted] を使用して測定されます。これらの PMC イベントは、OS とユーザー モードの両方でカウントされます。
	CPI	命令あたりのサイクル数 (CPI) は、IPC メトリクスの逆数です。これは、キャッシュミス、分岐予測ミス、メモリレイテンシ、その他のボトルネックにより、アプリケーションの実行にどう影響が及ぶかを示す基本的なパフォーマンスメトリクスの 1 つです。CPI 値が低いほど、パフォーマンスは良くなります。
	Branch Mis-prediction Ratio	予測ミスした分岐とリタイアした分岐命令の割合。
fp	Retired SSE/AVX Flops (GFLOPs)	リタイアした SSE/AVX FLOPS の数。
	FP Dispatch Faults (PTI)	浮動小数点命令のディスパッチフォルト。このメトリクスの単位は、1,000 命令あたり (PTI) です。
l1	IC (32B) Fetch Miss Ratio	命令キャッシュのフェッチミス率。
	Op Cache Fetch Miss Ratio	オペレーションキャッシュ (64B) のフェッチミス率。
	IC Access (PTI)	命令キャッシュアクセス (PTI)。
	IC Miss (PTI)	命令キャッシュミス (PTI)。
	DC Access (PTI)	すべてのデータキャッシュ (DC) アクセス。このメトリクスの単位は PTI です。

表 5. AMD EPYC™ “Zen 4” のパフォーマンス メトリクス (続き)

メトリクス グループ	メトリクス	説明
l2	L2 Access	すべての L2 キャッシュ アクセス。このメトリクスの単位は PTI です。
	L2 Access from IC Miss	IC ミスからの L2 キャッシュ アクセス。このメトリクスの単位は PTI です。
	L2 Access from DC Miss	DC ミスからの L2 キャッシュ アクセス。このメトリクスの単位は PTI です。
	L2 Access from HWPf	L2 ハードウェア プリフェッチからの L2 キャッシュ アクセス。このメトリクスの単位は PTI です。
	L2 Miss	すべての L2 キャッシュ ミス。このメトリクスの単位は PTI です。
	L2 Miss from IC Miss	IC ミスからの L2 キャッシュ ミス。このメトリクスの単位は PTI です。
	L2 Miss from DC Miss	DC ミスからの L2 キャッシュ ミス。このメトリクスの単位は PTI です。
	L2 Miss from HWPf	L2 ハードウェア プリフェッチからの L2 キャッシュ ミス。このメトリクスの単位は PTI です。
	L2 Hit	すべての L2 キャッシュ ヒット。このメトリクスの単位は PTI です。
	L2 Hit from IC Miss	IC ミスからの L2 キャッシュ ヒット。このメトリクスの単位は PTI です。
	L2 Hit from DC Miss	DC ミスからの L2 キャッシュ ヒット。このメトリクスの単位は PTI です。
	L2 Hit from HWPf	L2 ハードウェア プリフェッチからの L2 キャッシュ ヒット。このメトリクスの単位は PTI です。

表 5. AMD EPYC™ “Zen 4” のパフォーマンス メトリクス (続き)

メトリクス グループ	メトリクス	説明
tlb	L1 ITLB Miss	L1 命令変換ルックアサイド バッファ (ITLB) はミスとなったが、L2-ITLB とページ テーブル ウォークによる ITLB リロードではヒットした命令フェッチ。テーブル ウォーク要求の対象は、L1-ITLB ミスと L2-ITLB ミスです。このメトリクスの単位は PTI です。
	L2 ITLB Miss	L1-ITLB ミスと L2-ITLB ミスに起因したページ テーブル ウォークによる ITLB リロードの数。このメトリクスの単位は PTI です。
	L1 DTLB Miss	ロード/ストアのマイクロオペレーションによる L1 データ変換ルックアサイド バッファ (DTLB) ミスの数。このイベントでは、L2-DTLB ヒットと L2-DTLB ミスの両方がカウントされます。このメトリクスの単位は PTI です。
	L2 DTLB Miss	ロード/ストアのマイクロオペレーションによる L2 データ変換ルックアサイド バッファ (DTLB) ミスの数。このメトリクスの単位は PTI です。
	All TLBs Flushed	フラッシュされたすべての TLB。
l3	L3 Access	L3 キャッシュ アクセス。このメトリクスの単位は PTI です。
	L3 Miss	L3 キャッシュ ミス。このメトリクスの単位は PTI です。
	L3 Miss (%)	L3 キャッシュ ミスの割合。このメトリクスの単位は PTI です。
	Ave L3 Miss Latency	コア サイクルでの平均 L3 ミス レイテンシ。
Memory	Total Memory Bw (GB/s)	読み出しおよび書き込みの合計メモリ帯域幅。
	Local DRAM Read Data Bytes (GB/s)	ローカル プロセッサの DRAM 読み出しおよび書き込みデータ バイト。
	Local DRAM Write Data Bytes (GB/s)	
	Remote DRAM Read Data Bytes (GB/s)	リモート プロセッサの DRAM 読み出しおよび書き込みデータ バイト。
	Remote DRAM Write Data Bytes (GB/s)	
	Mem Ch-A RdBw (GB/s) Mem Ch-A WrBw (GB/s) ...	全チャネルのメモリ読み出しおよび書き込み帯域幅 (GB/s)。

表 5. AMD EPYC™ “Zen 4” のパフォーマンス メトリクス (続き)

メトリクス グループ	メトリクス	説明
xgmi	Local Inbound Read Data Bytes (GB/s)	CPU へのローカル受信データ バイト (例: 読み出しデータ)。
	Local Outbound Write Data Bytes (GB/s)	CPU からのローカル送信データ バイト (例: 書き込みデータ)。
	Remote Inbound Read Data Bytes (GB/s)	CPU へのリモート ソケット受信データ バイト (例: 読み出しデータ)。
	Remote Outbound Write Data Bytes (GB/s)	CPU からのリモート ソケット送信データ バイト (例: 書き込みデータ)。
	xGMI Outbound Data Bytes (GB/s)	合計送信データ バイト (1 秒あたりのギガバイト数)。
dma (AMD “Zen1”、 AMD “Zen2”、 AMD “Zen3” プロセッサでは 使用不可)	Total Upstream DMA Read Write Data Bytes (GB/s)	読み出しと書き込みを含む合計のアップストリーム DMA。
	Local Upstream DMA Read Data Bytes (GB/s)	ローカル アップストリーム DMA の読み出しデータ バイト。
	Local Upstream DMA Write Data Bytes (GB/s)	ローカル アップストリーム DMA の書き込みデータ バイト。
	Remote Upstream DMA Read Data Bytes (GB/s)	リモート ソケット アップストリーム DMA の読み出しデータ バイト。
	Remote Upstream DMA Write Data Bytes (GB/s)	リモート ソケット アップストリーム DMA の書き込みデータ バイト。
pcie	PCIe0 (GB/s) PCIe1 (GB/s) PCIe2 (GB/s) PCIe3 (GB/s)	PCIe 帯域幅 (概算値、GB/s)。

表 5. AMD EPYC™ “Zen 4” のパフォーマンス メトリクス (続き)

メトリクス グループ	メトリクス	説明
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)	各種ノードおよび CCX からのソフトウェア プリフェッチ データ キャッシュ。
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)	各種ノードおよび CCX からのハードウェア プリフェッチ データ キャッシュ。

表 5. AMD EPYC™ “Zen 4” のパフォーマンス メトリクス (続き)

メトリクス グループ	メトリクス	説明
pipeline_util	Total_Dispatch_Slots	1 サイクルで最大 6 命令をディスパッチ可能。
	SMT_Disp_contention	その他のスレッドが選択されたために使用されなかったディスパッチ スロットの割合。
	Frontend_Bound	フロントエンドで十分な命令/オペレーションが供給されなかったために未使用のままだったディスパッチ スロットの割合。
	Bad_Speculation	その他のスレッドが選択されたために使用されなかったディスパッチ スロットの割合。
	Backend_Bound	バックエンド ストールのために未使用のままだったディスパッチ スロットの割合。
	Retiring	リタイアしたオペレーションによって使用されたディスパッチ スロットの割合。
	IPC	サイクルあたりの命令。
	Frontend_Bound.Latency	命令キャッシュまたは ITLB ミスなど、フロントエンドでのレイテンシ ボトルネックのために未使用のままだったディスパッチ スロットの割合。
	Frontend_Bound.BW	デコード 帯域幅またはオペレーション キャッシュ フェッチ 帯域幅など、フロントエンドでの帯域幅 ボトルネックのために未使用のままだったディスパッチ スロットの割合。
	Bad_Speculation.Mispredicts	分岐予測 ミスのためにフラッシュされたディスパッチ オペレーションの割合。
	Bad_Speculation.Pipeline_Restarts	パイプラインの再開 (再同期) のためにフラッシュされたディスパッチ オペレーションの割合。
	Backend_Bound.Memory	メモリ サブシステムに起因するストールのために未使用のままだったディスパッチ スロットの割合。
	Backend_Bound.CPU	メモリ サブシステムに関連しないストールのために未使用のままだったディスパッチ スロットの割合。
	Retiring.Fastpath	リタイアした fastpath オペレーションによって使用されたディスパッチ スロットの割合。
	Retiring.Microcode	リタイアしたマイクロコード オペレーションによって使用されたディスパッチ スロットの割合。

2.3 コマンド

次の表にコマンドの一覧を示します。

表 6. AMDuProfPcm のオプション

コマンド	説明
roofline	ルーフライン モデルの生成に必要なデータを収集します。

2.4 例

2.4.1 Linux と FreeBSD

- 60 秒間にわたり、コア 0 から IPC データを収集します。
./AMDuProfPcm -m ipc -c core=0 -d 60 -o /tmp/pcmdata.csv
- 60 秒間にわたり、CCX=0 の IPC/L3 メトリクスを収集します。
./AMDuProfPcm -m ipc,l3 -c ccx=0 -d 60 -o /tmp/pcmdata.csv
- 60 秒間にわたり、すべての UMC でメモリ帯域幅のみを収集し、/tmp/pcmdata.csv ファイルに出力を保存します。
./AMDuProfPcm -m memory -a -d 60 -o /tmp/pcmdata.csv
- 60 秒間にわたり、すべてのコアから IPC データを収集します。
./AMDuProfPcm -m ipc -a -d 60 -o /tmp/pcmdata.csv
- コア 0 から IPC データを収集し、コア 0 でプログラムを実行します。
./AMDuProfPcm -m ipc -c core=0 -o /tmp/pcmdata.csv -- /usr/bin/taskset -c 0 <application>
- コア 0 ~ 7 から IPC データを収集し、コア 0 ~ 3 でアプリケーションを実行します。
./AMDuProfPcm -m ipc -c core=0-7 -o /tmp/pcmdata.csv -- /usr/bin/taskset -c 0-3 <application>
- コア 0 から IPC および I2 データを収集し、(時系列ではなく) 累積をレポートし、コア 0 でプログラムを実行します。
./AMDuProfPcm -m ipc,l2 -c core=0 -o /tmp/pcmdata.csv -C -- /usr/bin/taskset -c 0 <application>
- サポートされる生のコア PMC イベント リストを出力します。
./AMDuProfPcm -l
- 指定されたイベントの名前、説明、使用可能なユニット マスクを出力します。
./AMDuProfPcm -z pmcx03
- ルート モードでルーフライン データを収集します。
sudo ./AMDuProfPcm roofline -o /tmp/roofline.csv <application>
- 非ルート モードでルーフライン データを収集します。
./AMDuProfPcm roofline -X -o /tmp/roofline.csv <application>
- ルーフライン データをプロットし、出力ディレクトリ /tmp に PDF を生成します。
AMDuProfModelling.py -i /tmp/roofline.csv -o /tmp/

2.4.2 Windows

コア メトリクス

- サポートされるメトリクスのリストを取得します。

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

- 30 秒間にわたり、コア 0 から IPC データを収集します。

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

- 30 秒間にわたり、CCX=0 に含まれるすべてのコアの IPC/L2 メトリクスを収集します。

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

- 30 秒間にわたり、システム内すべてのコアから IPC データを収集します。

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

- コア 0 から IPC データを収集し、プログラムを実行します。

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

- すべてのコアから IPC および L2 データを収集し、システムおよびパッケージ レベルで集計したデータをレポートします。

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

- CCX=0 に含まれるすべてのコアから IPC および L2 データを収集し、(時系列ではなく) 累積をレポートします。

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

- すべてのコアから IPC および L2 データを収集し、(時系列ではなく) 累積をレポートします。

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

- すべてのコアから IPC および L2 データを収集し、(時系列ではなく) 累積をレポートし、システムおよびパッケージ レベルで集計します。

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

L3 メトリクス

- 30 秒間にわたり、ccx=0 から L3 データを収集します。

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

- 30 秒間にわたり、すべての CCX から L3 データを収集し、レポートします。

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

- 30 秒間にわたり、すべての CCX から L3 データを収集し、システムおよびパッケージ レベルで集計し、レポートします。

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

- 30 秒間にわたり、すべての CCX から L3 データを収集し、システムおよびパッケージ レベルで集計し、レポートします。個別の CCX に対してもレポートします。

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

- 30 秒間にわたり、すべての CCX から L3 データを収集し、累積データをレポートします (時系列データなし)。

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

- すべての CCX から L3 データを収集し、システムおよびパッケージ レベルで集計し、累積データをレポートします (時系列データなし)。

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

- 30 秒間にわたり、コア 0 から IPC データを収集します。

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

メモリ帯域幅

- 60 秒間にわたってすべてのメモリ チャネルのメモリ帯域幅をレポートし、c:\tmp\pcmdata.csv ファイルに出力を保存します。

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

- 60 秒間にわたって、システム レベルで集計された合計メモリ帯域幅をレポートし、c:\tmp\pcmdata.csv ファイルに出力を保存します。

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

- システム レベルで集計された合計メモリ帯域幅をレポートし、各メモリ チャネルについてもレポートします。

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

- システム レベルで集計された合計メモリ帯域幅をレポートし、すべての使用可能なメモリ チャネルについてもレポートします。時系列データの代わりに累積メトリクス値をレポートするには、次のコマンドを使用します。

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

生のイベント カウント ダンプ

- コア 0 からイベントを監視し、すべてのサンプルの生イベント カウントを時系列にダンプします。メトリクス レポートは生成されません。

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

- すべてのコアからイベントを監視し、すべてのサンプルの生イベント カウントを時系列にダンプします。メトリクス レポートは生成されません。

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

カスタム設定ファイル

サンプルの設定 XML ファイルが、<uprof-install-dir>\bin\Data\Config\SamplePcm-core.conf にあります。このファイルをコピーし、ユーザーごとに興味のある特定イベントとメトリクスの計算式に合わせて内容を変更できます。ファイル内に定義されたすべてのメトリクスが監視され、レポートされます。

```
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
```

その他

- サポートされる生のコア PMC イベント リストを出力します。

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

- 指定されたイベントの名前、説明、使用可能なユニット マスクを出力します。

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

2.5 BIOS 設定 - 既知の動作

次に、BIOS 設定に基づく L2 Hit from HWPF/L2 Miss from HWPF メトリクスの既知の動作を示します。

- BIOS で次のオプションがすべて無効になっている場合、AMDuProfPcm の L2 Hit from HWPF/L2 Miss from HWPF メトリクスはデータを一切収集しません。

- L1 Stream HW Prefetcher
- L1 Stride Prefetcher
- L1 Region Prefetcher
- L2 Stream HW Prefetcher
- L2 up/Down Prefetcher

- 次の BIOS 設定の場合、AMDuProfPcm の L2 Hit from HWPF/L2 Miss from HWPF メトリクスは非常に少ないサンプルしか収集しません。

- 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 ルート権限なしでの監視

Linux で、「msr」モジュールとルート アクセスに依存することなくメトリクスを監視するには、-x オプションを使用します。このオプションは、AMD “Zen” ベースのプロセッサで、コア、L3、DF の PMC イベントを収集します。最近のプロセッサでは、最新カーネルのサポートが必要になる場合があります。

例

- ベンチマークの IPC を時系列で監視し、スレッドあたりのメトリクスを集計します。

```
$ AMDuProfPcm -X -m ipc -o /tmp/pcm.csv -- /tmp/myapp.exe
```

- ベンチマークの IPC を時系列で監視し、プロセッサ パッケージあたりのメトリクスを集計します。

```
$ AMDuProfPcm -X -m ipc -A package -o /tmp/pcm.csv -- /tmp/myapp.exe
```

- ベンチマークの IPC を時系列で監視し、システム レベルでメトリクスを集計します。

```
$ AMDuProfPcm -X -m ipc -A system -o /tmp/pcm.csv -- /tmp/myapp.exe
```

- ベンチマーク実行の最後に、IPC メトリクスの累積をレポートします。

```
$ AMDuProfPcm -X -m ipc -C -o /tmp/pcm.csv -- /tmp/myapp.exe
```

- ベンチマーク実行の最後に、IPC メトリクスの累積をレポートし、プロセッサ パッケージごとにメトリクスを集計します。

```
$ AMDuProfPcm -X -m ipc -C -A package -o /tmp/pcm.csv -- /tmp/myapp.exe
```

- ベンチマーク実行の最後に、IPC メトリクスの累積をレポートし、システム レベルでメトリクスを集計します。

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

- メモリ帯域幅を時系列で監視し、パッケージおよびメモリ チャネル レベルでレポートします。

```
$ AMDuProfPcm -X -m memory -a -A system,package -o /tmp/mem.csv
```

- レベル 1 およびレベル 2 のトップダウン メトリクス (パイプライン使用率) を時系列で監視します。

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

- レベル 1 およびレベル 2 のトップダウン メトリクス (パイプライン使用率) の累積をレポートします。

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

トップダウンの結果を改善するには、NMI ウォッチドッグを無効にして、次のコマンドを root として実行します。

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

2.7 ルーフライン モデル

AMDuProfPcm は、アプリケーション パフォーマンスをメモリ トラフィックと浮動小数点計算のピークに関連付ける、基本的なルーフライン モデリングを提供します。このビジュアル パフォーマンス モデルは、浮動小数点演算向け並列ソフトウェアの改善に関するインサイトを提供します。アプリケーションの特性を評価し、ベンチマークがメモリまたは演算のいずれによって制限されているのかを識別するのに役立ちます。

このツールは、プロファイリング対象アプリケーションの実行中にメモリ トラフィックと浮動小数点演算を監視します。また、演算強度として「DRAM トラフィック 1 バイトあたりの演算数 [FLOPS/バイト]」を算出します。ルーフライン チャートは次のようにプロットされます。

- X 軸: 対数スケールでの (AI) 演算強度 (FLOPS/バイト)
- Y 軸: 対数スケールでのスループット (GFLOPS/秒)
- 水平ラインは、そのシステムの理論上のピーク浮動小数点パフォーマンス (ハードウェア制限) を表します。
- 斜めのラインは、ピーク メモリ パフォーマンスを表します。このラインは、次の式を使用してプロットされます。スループット = $\min$ (理論上のピーク GFLOPS/秒、理論上のピーク メモリ帯域幅 * AI)。

このツールはデフォルトで、次の水平ルーフラインをプロットします。

- 単精度浮動小数点ピーク (「SP FP ピーク」)
- 倍精度浮動小数点ピーク (「DP FP ピーク」)

最大ピークの水平 (計算) ピーク ルーフラインをプロットするのに使用できるオプションは、次のとおりです。

- 単精度 noSIMD および noFMA
- 倍精度 noSIMD および noFMA

次の手順で、アプリケーションのルーフライン チャートを生成します。

1. AMDuProfPcm を使用して、プロファイル データを収集します。

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

AMD “Zen4” 9xx4 シリーズ プロセッサで、Linux カーネルが DF カウンターのアクセスをサポートしていない場合は、ルート権限を使用して次のコマンドを実行します。

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

2. ルーフライン チャートを生成するには、次のコマンドを実行します。

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

ルーフライン チャートは、*/tmp/AMDuProf_roofline-2022-10-28-19h00m10s.pdf* ファイルに保存されます。

ルーフライン チャートを生成する際のヒントは次のとおりです。

- データの収集中、AMDuProfPcm が非ルート権限で起動された場合は、-memspeed オプションを使用して DRAM 速度を指定します。dmidecode または lshw コマンドを使用すると、メモリ速度を取得できます。
- 追加の計算水平ピーク ラインをプロットするには、次のオプションを使用します。

- --sp-roofs: 単精度 noSIMD および noFMA の最大ピーク ルーフをプロットします。

- --dp-roofs: 倍精度 noSIMD および noFMA の最大ピーク ルーフをプロットします。

例:

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

- グラフ チャート内に出力するアプリケーション名を指定するには、-a <appname> オプションを使用します。
- このツールは、メモリ トラフィックと浮動小数点パフォーマンスに対して理論上の最大ピークを使用します。このため、ピーク メモリ帯域幅を取得するには STREAM などのベンチマークを使用し、ピーク FLOPS については HPL または GEMM を使用できます。これらのスコアを使用して、ルーフライン チャートをプロットできます。次のオプションを使用します。

- --stream <STREAM score>

- --hpl <HPL score>

- --gemm <SGEMM | DGEMM score>

ルーフライン チャートのサンプルを次に示します。

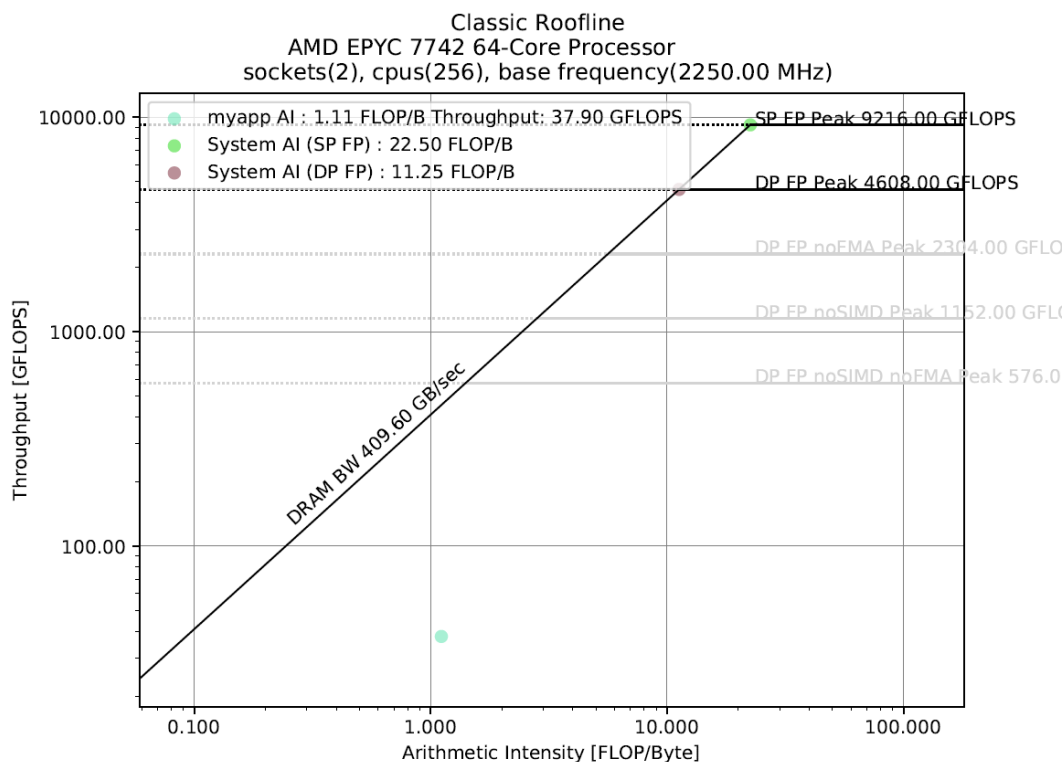


図 1. サンプルルーフライン チャート

2.8 パイプライン使用率

AMD “Zen4” ベースのプロセッサで、AMDuProfPcm は、パイプライン使用率 (pipeline_util) メトリクスの監視とレポートをサポートしています。この機能は pipeline_util メトリクスを提供して、CPU パイプライン内のボトルネックを可視化します。レベル 1 およびレベル 2 のトップダウン メトリクスを監視およびレポートするには、-m pipeline_util オプションを使用します。

レベル 1 メトリクスは次のとおりです。

表 7. レベル 1 メトリクス

メトリクス	説明
Total_Dispatch_Slots	合計ディスパッチ スロット。1 サイクルで最大 6 命令をディスパッチ可能。
SMT_Dispatch_contention	その他のスレッドが選択されたために未使用だったディスパッチ スロット。
Frontend_Bound	フロントエンドで適切な命令/オペレーションが供給されなかったために未使用のままだったディスパッチ スロット。
Bad_Speculation	ディスパッチされたオペレーションで、リタイアしなかったもの。
Backend_Bound	バックエンド ストールのために未使用のままだったディスパッチ スロット。
Retiring	リタイアしたオペレーションによって使用されたディスパッチ スロット。

レベル 2 メトリクスは次のとおりです。

表 8. レベル 2 メトリクス

メトリクス	説明
Frontend_Bound.Latency	命令キャッシュまたは ITLB ミスなど、フロントエンドでのレイテンシボトルネックに起因する未使用ディスパッチ スロット。
Frontend_Bound.BW	デコード帯域幅またはオペレーション キャッシュ フェッチ帯域幅など、フロントエンドでの帯域幅ボトルネックに起因する未使用ディスパッチ スロット。
Bad_Speculation.Mispredicts	分岐予測ミスのためにフラッシュされたディスパッチ オペレーション。
Bad_Speculation.Pipeline_Restarts	パイプラインの再開 (再同期) のためにフラッシュされたディスパッチ オペレーション。
Backend_Bound.Memory	メモリ サブシステムに起因するストールのために未使用のままだったディスパッチ スロット。
Backend_Bound.CPU	メモリ サブシステムに関連しないストールのために未使用のままだったディスパッチ スロット。
Retiring.Fastpath	リタイアした fastpath オペレーションによって使用されたディスパッチ スロット。
Retiring.Microcode	リタイアしたマイクロコード オペレーションによって使用されたディスパッチ スロット。

多重化のために、レポートされるメトリクスが矛盾する場合があります。多重化による影響を最小限に抑えるには、-x オプションを使用します。より良い結果を得るために、タスクセットを使用して監視対象アプリケーションを特定のコア セットに関連付け、監視対象アプリケーションが実行されているコアだけを監視します。

トップダウン メトリクスを収集するには、次のコマンドを実行します。

```
$ 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
```

レポートのサンプルを次に示します。

Total_Dispatch_Slots	SMT_Dispatch_Slots	Frontend_Bound.Latency	Frontend_Bound.BW	Bad_Speculation.Mispredicts	Bad_Speculation.Pipeline_Restarts	Backend_Bound.Memory	Backend_Bound.CPU	Retiring.Fastpath	Retiring.Microcode
31528583352	0	0.18	0	33.19	66.39	3.85	0.18	0	0
31801404234	0	0.19	0.96	32.79	66.98	3.95	0.19	0	0.91 0.05
31876659852	0	0.18	0.22	32.73	67.53	3.95	0.18	0	0.2 0.02
31889169876	0	0.18	0.08	32.54	66.96	4.01	0.19	0	0.08 0
31914934404	0	0.19	0	32.76	66.92	3.97	0.19	0	0 0
31991329650	0	0.19	0	39.84	59.33	3.53	0.21	0	0 0
31851469458	0	0.19	0.22	59.47	40.21	2.36	0.22	0	0.04 0.18
31949156628	0.01	0.19	0.21	59.3	40.39	2.33	0.22	0	0.04 0.17
31817293530	0	0.19	0.6	59.15	40.39	2.38	0.22	0	0.1 0.5
31832631192	0	5.95	18.46	43.48	32.79	1.84	5.44	0.5	18.39 0.07
31912889772	0	10	31.77	31.46	26.5	1.48	9.21	0.79	31.75 0.02
31883125182	0	10.08	31.78	31.71	26.5	1.47	9.24	0.84	31.76 0.02
31993782744	0	10	30.81	31.72	26.47	1.45	9.15	0.85	30.79 0.02
31858516134	0	10	31.56	31.83	26.32	1.45	9.15	0.85	31.54 0.02
31928600622	0	9.95	31.78	31.85	26.49	1.43	9.03	0.92	31.76 0.02
31802799444	0	9.98	32.24	31.71	26.48	1.45	9.13	0.85	32.22 0.02
31827460368	0	5.4	17.59	38.25	40.03	2.25	5.04	0.37	17.56 0.03
31937807754	0	0.14	0.05	44.44	54.97	3.22	0.17	0	0.03 0.02
31865155098	0	0.15	0.07	44.79	54.97	3.22	0.17	0	0.04 0.03
31977071160	0	0.15	0	44.79	54.89	3.18	0.17	0	0 0

図 2. サンプル レポート

例

- シングルスレッド プログラムで、レベル 1 およびレベル 2 のトップダウン メトリクス (パイプライン使用率) を時系列で監視します。

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

- すべてのコアで実行されているマルチスレッド プログラムで、レベル 1 およびレベル 2 のトップダウン メトリクスを時系列で監視します。

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

- すべてのコアで実行されているマルチスレッド プログラムで、レベル 1 およびレベル 2 のトップダウン メトリクスを累積で監視します。

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

第 3 章 AMDuProfSys の使用開始

3.1 概要

AMDuProfSys は、python ベースの AMD プロセッサ向けシステム解析ツールです。ハードウェア イベントを収集し、収集された生のイベントを使用して、単純なカウンター値または複雑なレシピを評価するために使用できます。パフォーマンス メトリクスは、コア、L3、DF、UMC の PMC を使用して収集されるプロファイルデータに基づきます。このツールを使用して、システム内で使用されているハードウェア ブロックのパフォーマンスの詳細を全般的に取得できます。

3.2 サポートされるプラットフォーム

AMDuProfSys は、次のバリエーションの AMD EPYC™ 7002、7003、9000 シリーズ プロセッサをサポートします。

- ファミリ 17、モデル 0x30 - 0x3F
- ファミリ 19、モデル 0x0 - 0xF
- ファミリ 19、モデル 0x1 - 0x1F
- ファミリ 19、モデル 0x20 - 0x2F
- ファミリ 19、モデル 0xA0 - 0xAF

3.3 サポートされるハードウェア カウンター

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

3.4 サポートされるオペレーティング システム

- Linux
- Windows

3.5 セットアップ

5 ページの「AMD uProf のインストール」のインストール手順に従います。

3.5.1 Linux

tar ボールが使用されている場合、uProf ドライバーを手動で使用する必要があります。uProf ドライバーを使用しない場合は、任意で Linux perf を使用できます。ただし、Linux ユーザー スペース ツールがインストールされており、perf ツールが必要な PMC イベントの監視をサポートしていることを確認する必要があります。uProf ドライバーが使用されていない場合、コマンド ラインに `--use-linux-perf` オプションを含める必要があります。

ユーザー スペース perf ツールをインストールするには、次のコマンドを実行します。

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

NMI ウォッチドッグを無効にする必要があります。これにはルート権限が必要です。

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

システム全体のプロファイル データ、または、DF および L3 メトリクスを収集する必要がある場合、perf パラメーターに `-1` を設定します。

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

3.5.2 Windows

セットアップ ファイルにより、AMDuProfSys の実行に必要なすべてのコンポーネントがインストールされます。

インストールの後で、次のディレクトリから AMDuProfSys を使用できます。

<インストール ディレクトリ>/bin/AMDPerf/AMDuProfSys.py

Python パッケージ

AMDuProfSys では、ターゲット プラットフォームに Python がインストールされている必要があります。サポートされる最小バージョンは、Python 3.6 です。ツールの初回実行時に、次の Python モジュールのインストールが要求されます。

- `tqdm` — `pip3 install tqdm` を使用してインストール
- `xlsxwriter` — `pip3 install xlsxwriter` を使用してインストール
- `yaml` — `apt-get install python-yaml` または `pip3 install pyyaml` を使用してインストール
- `yamlordereddictloader` — `pip3 install yamlordereddictloader` を使用してインストール
- `rich` — `pip3 install rich` を使用してインストール

書式

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

<OPTIONS> — 収集、レポート生成、またはこのツールのヘルプの表示

<PROGRAM> — プロファイリングの対象となる起動アプリケーションを指定

<ARGS> — 起動アプリケーションの引数のリストを指定

一般的な使用法:

- ヘルプを表示します。

```
AMDuProfSys.py -h
```

- デフォルト メトリクス (コア、L3、DF) を収集します。

```
AMDuProfSys.py -o default -a -d 100
```

- すべてのカウンターを同時に収集し、レポートします。

```
AMDuProfSys.py --config core,l3,df,umc -o all -a -d 100
```

- コマンド ラインからユーザーが定義したカスタム メトリクスを収集します。

```
./AMDuProfSys.py --metrics core/BrMisPredExTime="(0x4300C3)/(0x4300C2)",core/  
ratio="((BrMisPredExTime * 0x430076)/0x4300C0)" -d 20
```

- コア 0 からコア メトリクスを収集し、コアでアプリケーションを実行します。

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

- 収集中に生成されたセッション ファイルから .csv 形式のレポートを生成します。

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

- .xls 形式のレポートを生成します。

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

- コア メトリクス (コア 0 ~ 5) の時系列プロファイル データを 1000 ms の間隔で収集し、実行中アプリケーションのアフィニティをコア 0 に設定します。

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

注記: 時系列プロファイル データを収集できるのは、-use-linux-perf オプションを使用した場合のみです。

- コア、L3、DF、UMC メトリクスを同時に収集します。

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

3.6 オプション

3.6.1 一般

次の表に一般的なオプションの一覧を示します。

表 9. AMDuProfSys の一般的なオプション

オプション	説明
-h, --help	使用法の表示
-v, --version	バージョンの出力
--system-info	システム情報
--enable-irperf	irperf の有効化 注記: Linux のみで使用でき、ルート権限が必要です。
--mux-interval-core <ms>	多重化の間隔をミリ秒で設定
--mux-interval-l3 <ms>	多重化の間隔をミリ秒で設定
--mux-interval-df <ms>	多重化の間隔をミリ秒で設定、デフォルトは 4 ms

3.6.2 収集コマンド

次の表に、収集コマンドのオプションの一覧を示します。

表 10. AMDuProfSys 収集コマンドのオプション

オプション	説明
--config	指定されたアプリケーションを起動し、生のイベントを監視します。パスを使用して、事前定義した一連の設定ファイルまたは単一設定ファイルを使用するように、収集コマンドを設定できます。
-a, --all-cpus	すべてのコアから収集します。 注記: オプション -c と -a は同時に使用できません。
-C, --cpu <CPUs>	監視する CPU のリスト。スペースなしのカンマ区切りリストで、複数の CPU を指定します。例: 0,1。 CPU の範囲を指定します。例: 5-10。
-d, --duration <seconds>	実行するプロファイリングの長さ。 注記: 起動アプリケーションが指定された場合は正しく機能しません。
-t, --tid <tid>	既存スレッドのイベントを監視します。カンマ区切りリストで、複数の TID を指定できます。 注記: Linux のみで使用できます。
-p, --pid <pid>	既存プロセスのイベントを監視します。カンマ区切りリストで、複数の PID を指定できます。 注記: Linux のみで使用できます。
-o, --output <file>	生のイベント カウント 値を保存する出力ファイルの名前。
--no-inherit	子タスクが監視されなくなります。 注記: Linux のみで使用できます。

表 10. AMDuProfSys 収集コマンドのオプション (続き)

オプション	説明
-I, --interval <n>	生のイベント カウントの詳細がファイルに保存される間隔。時系列データの収集では必須です。 注記: Linux のみで使用できます。
-V, --verbose	詳細なデータを出力します。
-r --collect-raw	生のイベント ファイルを使用してイベントを収集します。レポートを生成できるのは AMD のみです。このオプションは、一連の大量のメトリクスを収集する際に役立ちます。
--use-linux-perf	このオプションを使用すると、Linux で、AMDuProf ドライバーの代わりに Linux Perf を使用して、プロファイル データを収集できます。
-m --metrics	コマンド ラインを介してユーザーが定義したカスタム メトリクスを収集します。
--affinity	CPU のカンマ区切りリスト。指定された CPU 上でワークロードが実行されます。

3.6.3 レポート コマンド

計算されたメトリクスを含むプロファイルレポートを生成します。収集コマンドにより、.ses 拡張子付きのプロファイル セッション ファイルと、プロファイル収集タイプ別の生カウンター データ ファイルが生成されます。次のコマンド オプションに示すとおり、レポートを生成するには、-i オプションを使用してセッション ファイルを指定する必要があります。

表 11. AMDuProfSys レポート コマンドのオプション

オプション	説明
-i, --input-file <file>	収集コマンドによって生成されたセッション ファイルを入力します。
--config <file>	設定ファイルまたはオプション コア/df/l3 を使用して、イベント セットとメトリクスを指定します。
-o, --output <file>	設定に従った .csv または .xls 形式の出力ファイルの名前。
-f, --format	出力ファイルの形式 (.xls または .csv)。デフォルトのファイル形式は .csv です。
--group-by <system, package, numa, ccx>	選択されたグループに基づいて結果を集計します。デフォルトは none です。
-T, --time-series	コアごとの時系列レポートを生成します。サポートされるのは .csv 形式のみです。時系列データを生成するには、収集時に -I オプションを指定する必要があります。 注記: Linux のみで使用できます。
--set-precision <n>	レポートされるメトリクスの浮動小数点精度を設定します。デフォルト値は 2 です。
-V, --verbose	詳細なデータを出力します。

3.7 例

- システム全体を監視して、設定ファイルに定義されたメトリクスを収集および生成し、プロファイル レポートを生成します。

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

- コアのアフィニティをコア 0 に設定してプログラムを起動し、そのコアを監視してプロファイル レポートを生成します。

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

- プログラムを起動して監視し、プロファイル レポートを生成します。

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

注記: 処理を多重化する場合、-a または -C オプションが必須です。

- 2 つのステップでレポートを収集し、生成します。
 - 生のイベント カウント値を含むバイナリ データファイル *sci_perf.data* を生成します。

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

- 計算されたメトリクスを含むレポート ファイル *sci_perf.csv* を生成します。

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

- 2 つのステップで、複数の設定ファイルを使用して収集し、レポートを生成します。

- 生のイベント カウント値を含むバイナリ データファイル *sci_perf.data* を生成します。

```
AMDuProfSys.py collect --config core,13,df -C 0-10 -o sci_perf taskset -c 0 scimark2
```

注記: -C、-a オプションを使用できるのは、コア カウンターの場合のみです。

- 計算されたメトリクスを含むレポート ファイル *sci_perf.csv* を生成します。

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

- 多重化の間隔を更新します。

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

注記: --mux-interval-core オプションを使用するには、ルート アクセスが必要です。

3.8 制限

- 次のプラットフォームの場合、Linux では UMC プロファイリングを使用できません。
 - ファミリ 17、モデル 0x30 - 0x3F
 - ファミリ 19、モデル 0x0 - 0xF
- 時系列プロファイル データを収集できるのは、Linux で --use-linux-perf オプションを使用する場合のみです。

パート 3: アプリケーション解析

第 4 章 ワークフローと主な概念

4.1 ワークフロー

AMD uProf のワークフローには、次のフェーズがあります。

1. 収集 — アプリケーション プログラムを実行し、プロファイル データを収集します。
2. 変換 — プロファイル データを処理して、集計、相関付けし、体系化してデータベース内に格納します。
3. 解析 — パフォーマンス データを表示して解析し、ボトルネックを特定します。

4.1.1 収集フェーズ

このセクションでは、収集フェーズの重要な概念について説明します。

プロファイル ターゲット

プロファイル ターゲットは次のうちのいずれかで、そのターゲットに対してプロファイル データが収集されます。

- アプリケーション — アプリケーションを起動して、そのプロセスおよび子プロセスをプロファイリングします。
- システム — 実行中のすべてのプロセスおよび/またはカーネルをプロファイリングします。
- プロセス — 実行中のアプリケーションに関連付けます (ネイティブ アプリケーションのみ)。

プロファイル タイプ

プロファイル タイプにより、収集されるプロファイル データの種類とデータ収集方法が定義されます。サポートされるプロファイル タイプは次のとおりです。

- CPU プロファイル
- CPU トレース
- GPU プロファイル
- GPU トレース
- システム全体の消費電力プロファイル

データ収集は、**サンプリング設定**によって定義されます。

- **サンプリング設定**により、一連のサンプリング イベントとそのサンプリング間隔、モードが識別されます。
- **サンプリング イベント**は、サンプリング ポイントをトリガーするために使用されるリソースです。サンプル (プロファイル データ) は、サンプリング ポイントで収集されます。
- **サンプリング間隔**として定義されたサンプリング イベント数の発生後、サンプル収集のための割り込みが生成されます。

- モードにより、いつサンプリング イベントの発生をカウントするのかが決まります (ユーザー モードおよび/または OS モード)。

サンプル対象データは収集されるプロファイル データの種類で、次のように定義されます。

サンプル対象データ — サンプリング イベントのサンプリング間隔に達して、割り込みが生成されたときに収集されるプロファイル データ。

次の表に、収集されるプロファイル データの種類と、プロファイル タイプ別のサンプリング イベントを示します。

表 12. サンプル対象データ

プロファイル タイプ	収集されるプロファイル データの種類	サンプリング イベント
CPU プロファイル	• プロセス ID • スレッド ID • IP • コールスタック • ETL トレース (Windows のみ) • OpenMP トレース — OMPT (Linux) • MPI トレース — PMPI (Linux) • OS トレース — Linux BPF	• OS タイマー • コア PMC イベント • IBS
CPU トレース	• ユーザー モード トレース — syscall および pthread データを収集 • OS トレース — スケジュール、diskio、syscall、pthread、funccount データを収集	なし
GPU プロファイル	Perfmon メトリクス	なし
GPU トレース	ランタイム トレース — HIP および HSA	なし

CPU プロファイルの場合、監視できるマイクロアーキテクチャ固有イベントが非常に多くあります。このツールによって、監視している、関連のある関心イベントが、**事前定義サンプリング設定**としてグループ化されます。たとえば、**Assess Performance** は事前定義サンプリング設定の 1 つであり、全般的なパフォーマンス評価を取得し、調査のために潜在的な問題を見つけるために使用できます。詳細は、[46 ページの「事前定義ビュー設定」](#)を参照してください。

カスタム サンプリング設定には、ユーザーが関心のあるイベントを含むサンプリング設定を定義できます。

プロファイル設定

プロファイル設定により、測定値の収集に使用されるすべての情報が識別されます。ここには、プロファイル ターゲット、サンプリング設定、サンプル対象データ、プロファイル スケジューリングの詳細に関する情報が含まれます。

GUI には、こういったプロファイル設定情報がデフォルト名 (例: AMDuProf-TBP-Classic) で保存されていますが、ユーザーが新たに定義することもできます。パフォーマンス解析は反復作業ですが、この設定は永続的であるため (削除も可能)、将来的にデータ収集を実行する際、同じ設定を再利用できます。

プロファイル セッション (またはプロファイル実行)

プロファイル セッションは、1 つのプロファイル設定に対する 1 回のパフォーマンス試験です。ツールにより、すべてのプロファイルおよび変換済みデータ (データベース形式) が、<プロファイル設定名>-<タイムスタンプ> という名前のフォルダーに保存されます。

プロファイル データは収集された後、uProf でのデータ処理により集計され、サンプル特性の原因となるプロセス、スレッド、ロード モジュール、関数、および命令が特定されます。その後、この集計済みデータは、解析フェーズで使用される SQLite データベースに書き込まれます。生プロファイル データを変換するこのプロセスは、CLI のプロファイル レポート 生成または GUI の視覚化生成時に実行されます。

4.1.2 変換およびレポート プロセス

収集された生のプロファイル データが集計され、原因となるプロセス、スレッド、ロード モジュール、関数、および命令が特定されます。サンプルを関数およびソース行に関連付けるには、コンパイラによって生成される起動アプリケーションのデバッグ情報が必要です。

このフェーズは、プロファイリングが停止された後、GUI で自動的に実行されます。CLI では、レポート コマンドによって、生プロファイル データが暗黙的に処理 (変換) され、CSV 形式のレポートが生成されます。CLI では、変換のみを実行する変換コマンドも提供されており、変換済みデータ ファイルを視覚化のために GUI にインポートできます。

4.1.3 解析フェーズ

ビューの設定

ビューとは、イベント データと計算されたパフォーマンス メトリクスをまとめたもので、GUI ページか、CLI で生成されたテキスト レポートで表示できます。事前定義サンプリング設定ごとに、関連付けられた事前定義ビューのリストがあります。

ツールを使用すると、**事前定義ビュー**と呼ばれる特定の設定だけをフィルター /表示できます。たとえば、**IPC assessment** ビューでは、CPU クロック、リタイアした命令、IPC、CPI などのメトリクスがリスト表示されます。詳細は、[44 ページの「事前定義サンプリング設定」](#)を参照してください。

4.2 事前定義サンプリング設定

事前定義サンプリング設定は、プロファイル解析に役立つ一連のサンプリング イベントを選択するのに便利な方法を提供します。次の表に、事前定義サンプリング設定の一覧を示します。

表 13. 事前定義サンプリング設定

プロファイル タイプ	事前定義設定の名前	略語	説明
時間ベース プロファイル (TBP)	Time-based profile	tbp	プログラムが時間を消費している箇所を特定します。

表 13. 事前定義サンプリング設定 (続き)

プロファイル タイプ	事前定義設定の名前	略語	説明
イベント ベース プロファイル (EBP)	Assess performance	assess	全般的なパフォーマンス評価を提供します。
	Assess performance (Extended)	assess_ext	全般的なパフォーマンス評価と追加のメトリクスを提供します。
	Investigate data access	data_access	L1 データ キャッシュの局所性と DTLB 動作が不十分なデータ アクセス操作を特定します。
	Investigate instruction access	inst_access	L1 命令キャッシュの局所性と ITLB 動作が不十分な命令フェッチを特定します。
	Investigate branching	branch	予測が不十分な分岐および near リターンを特定します。
	Investigate CPI	cpi	実行中アプリケーションまたはシステム全体の CPI および IPC メトリクスを解析します。
	Threading Analysis	threading	全般的なスレッド解析を取得し、詳しい調査のために潜在的な問題を特定します。 注記: この設定は Linux のみで使用できます。AMD “Zen3” および AMD “Zen4” プロセッサのみでサポートされます。
IBS	Instruction based sampling	ibs	IBS フェッチと IBS オペレーションを使用して、サンプル データを収集します。サンプル特性の原因となる命令を正確に特定します。
	Cache Analysis	memory	キャッシュ ラインの偽共有の問題を特定します。BS オペレーションを使用して、プロファイル データを収集します。

注記:

1. AMDuProf GUI は、上記の表に示した事前定義設定の**名前**を使用します。
2. [44 ページの表 13](#) に示した略語は、AMDuProfCLI の **collect** コマンドの **--config** オプションで使用されます。
3. サポートされる事前定義設定とそこで使用されるサンプリング イベントは、プロセッサのファミリーとモデルによって異なります。

4.3 事前定義ビュー設定

ビューとは、サンプリングされたイベント データと計算されたパフォーマンス メトリクスをまとめたもので、GUI か、CLI で生成されたテキスト レポートで表示できます。事前定義サンプリング設定ごとに、関連付けられた事前定義ビューのリストがあります。

次に、**Assess Performance** の事前定義ビュー設定の一覧を示します。

表 14. Assess Performance の設定

ビューの設定	略語	説明
Assess Performance	triage_assess	このビューは、パフォーマンスの全体像を提供し、クロック サイクルあたりの実行命令数 (IPC)、データ キャッシュ アクセス/ミス、予測ミスした分岐、ミスアライン データ アクセスを含みます。詳しい調査に向けて潜在的な問題を見つけるために使用できます。
IPC assessment	ipc_assess	命令レベルの並列性が低いホットスポットを見つけます。パフォーマンス インジケータとして IPC と CPI を提供します。
Branch assessment	br_assess	このビューは、分岐密度が高く分岐予測は不十分なコードを見つけるために使用できます。
Data access assessment	dc_assess	DC ミス レートと DC ミス率を含むデータ キャッシュ (DC) アクセスに関する情報を提供します。
Misaligned access assessment	misalign_assess	これは、ミスアライン データにアクセスしているコード領域を特定するために使用できます。

次の表に、スレッド設定の一覧を示します。

表 15. スレッド設定

ビューの設定	略語	説明
IPC assessment	ipc_assess	命令レベルの並列性が低いホットスポットを見つけます。パフォーマンス インジケータとして IPC と CPI を提供します。 注記: この設定は Linux のみで使用できます。AMD “Zen3” および AMD “Zen4” プロセッサのみでサポートされます。

次の表に、**Investigate Data Access** の事前定義ビュー設定の一覧を示します。

表 16. Investigate Data Access の設定

ビューの設定	略語	説明
IPC assessment	ipc_assess	命令レベルの並列性が低いホット スポットを見つけます。パフォーマンス インジケータとして IPC と CPI を提供します。
Data access assessment	dc_assess	DC ミス レートと DC ミス率を含むデータ キャッシュ (DC) アクセスに関する情報を提供します。
Data access report	dc_focus	このビューは、L1 データ キャッシュ (DC) の動作を解析して、ミスとリフィルを比較するために使用できます。
Misaligned access assessment	misalign_assess	ミスアライン データにアクセスしているコード領域を特定します。
DTLB report	dtlb_focus	L1 DTLB アクセスとミス レートに関する情報を提供します。

次の表に、**Investigate Branch** の事前定義ビュー設定の一覧を示します。

表 17. Investigate Branch の設定

ビューの設定	略語	説明
Investigate Branching	Branch	このビューは、分岐密度が高く分岐予測は不十分なコードを見つけるために使用できます。
IPC assessment	ipc_assess	命令レベルの並列性が低いホット スポットを見つけます。パフォーマンス インジケータとして IPC と CPI を提供します。
Branch assessment	br_assess	このビューは、分岐密度が高く分岐予測は不十分なコードを見つけるために使用できます。
Taken branch report	taken_focus	このビューは、成立した分岐数が多いコードを見つけるために使用できます。
Near return report	return_focus	このビューは、near リターンの予測が不十分なコードを見つけるために使用できます。

次の表に、**Assess Performance (Extended)** の事前定義ビュー設定の一覧を示します。

表 18. Assess Performance (Extended) の設定

ビューの設定	略語	説明
Assess Performance (Extended)	triage_assess_ext	このビューは、パフォーマンスの全体像を提供します。詳しい調査に向けて潜在的な問題を見つけるために使用できます。
IPC assessment	ipc_assess	命令レベルの並列性が低いホット スポットを見つけます。パフォーマンス インジケータとして IPC と CPI を提供します。
Branch assessment	br_assess	このビューを使用して、分岐密度が高く分岐予測は不十分なコードを見つけます。
Data access assessment	dc_assess	DC ミスレートと DC ミス率を含むデータ キャッシュ (DC) アクセスに関する情報を提供します。
Misaligned access assessment	misalign_assess	ミスアライン データにアクセスしているコード領域を特定します。

次の表に、**Investigate Instruction Access** の事前定義ビュー設定の一覧を示します。

表 19. Investigate Instruction Access の設定

ビューの設定	略語	説明
IPC assessment	ipc_assess	命令レベルの並列性が低いホット スポットを見つけます。パフォーマンス インジケータとして IPC と CPI を提供します。
Instruction cache report	ic_focus	このビューは、命令キャッシュ (IC) をミスするコード領域を特定するために使用できます。
ITLB report	itlb_focus	このビューは、ITLB ミスレートを解析し、レベル L1 および L2 別に分類するために使用できます。

次の表に、**Investigate CPI** の事前定義ビュー設定の一覧を示します。

表 20. Investigate CPI の設定

ビューの設定	略語	説明
IPC assessment	ipc_assess	命令レベルの並列性が低いホット スポットを見つけます。 パフォーマンス インジケータとして IPC と CPI を提供します。

次の表に、**Instruction Based Sampling** の事前定義ビュー設定の一覧を示します。

表 21. Instruction Based Sampling の設定

ビューの設定	略語	説明
IBS fetch overall	ibs_fetch_overall	このビューは、IBS フェッチ サンプル データの総合的なサマリを表示するために使用できます。
IBS fetch instruction cache	ibs_fetch_ic	このビューは、IBS フェッチ試行命令キャッシュ (IC) ミス データのサマリを表示するために使用できます。
IBS fetch instruction TLB	ibs_fetch_itlb	このビューは、IBS フェッチ試行 ITLB ミスのサマリを表示するために使用できます。
IBS fetch page translations	ibs_fetch_page	このビューは、フェッチ試行の IBS L1 ITLB ページ変換のサマリを表示するために使用できます。
IBS All ops	ibs_op_overall	このビューは、すべての IBS オペレーション サンプルのサマリを表示するために使用できます。
IBS MEM all load/store	ibs_op_ls	このビューは、IBS オペレーションのデータ ロード/ストアのサマリを表示するために使用できます。
IBS MEM data cache	ibs_op_ls_dc	このビューは、IBS オペレーションのサンプル ロード/ストアから得られた DC 動作のサマリを表示するために使用できます。
IBS MEM data TLB	ibs_op_ls_dtlb	このビューは、IBS オペレーションのデータ ロード/ストアから得られた DTLB 動作のサマリを表示するために使用できます。
IBS MEM locked ops and access by type	ibs_op_ls_memacc	このビューは、キャッシュ不可能 (UC) メモリ アクセス、Write Combining (WC) メモリ アクセス、ロックされたロード/ストア オペレーションを表示するために使用できます。
IBS MEM translations by page size	ibs_op_ls_page	このビューは、DTLB アドレス変換のサマリをページ サイズ別に表示するために使用できます。
IBS MEM forwarding and bank conflicts	ibs_op_ls_expert	このビューは、メモリ アクセス バンクの競合、データ転送、Missed Address Buffer (MAB) ヒットを表示するために使用できます。
IBS BR branch	ibs_op_branch	このビューは、予測ミスした分岐と成立した分岐を含む、IBS のリタイア済み分岐オペレーションの測定値を表示するために使用できます。
IBS BR return	ibs_op_return	このビューは、リターン予測ミス率を含む、IBS のリターン オペレーションの測定値を表示するために使用できます。
IBS NB local/remote access	ibs_op_nb_access	このビューは、ローカルおよびリモート アクセスの数とレイテンシを表示するために使用できます。

表 21. Instruction Based Sampling の設定 (続き)

ビューの設定	略語	説明
IBS NB cache state	ibs_op_nb_cache	このビューは、NB キャッシュ サービス要求の Owned (O) および Modified (M) キャッシュ ステートを表示するために使用できます。
IBS NB request breakdown	ibs_op_nb_service	このビューは、NB アクセス要求の分類を表示するために使用できます。
AMD “Zen3” および AMD “Zen4” プロセッサの新しいビュー		
IBS Load Op Analysis	ibs_op_ld	このビューは、アプリケーションのメモリ ロード パフォーマンスの問題を解析するために使用できます。
IBS Load Op Analysis (ext)	ibs_op_ld_ext	このビューは、アプリケーションのメモリ ロード パフォーマンスの問題を解析するために使用できます。
IBS Branch Overview	mibs_op_br_overview	このビューは、分岐メトリクスを解析するために使用できます。
IBS Load Latency Analysis	ibs_op_ld_lat	このビューは、アプリケーションのメモリ ロード レイテンシ パフォーマンスの問題を解析するために使用できます。
IBS Memory Overview	ibs_op_ls_overview	このビューは、アプリケーションのメモリ アクセス パターンを理解するために使用できます。
IBS Perf Overview	ibs_op_overview	このビューは、アプリケーションのパフォーマンス特性を理解するために使用できます。

注記:

1. AMDuProf GUI は、上記の表に示した事前定義設定の「ビューの設定」の**名前**を使用します。
2. 略語は、CLI 生成レポート ファイルで使用されます。
3. サポートされる事前定義設定とそこで使用されるサンプリング イベントは、プロセッサのファミリーとモデルによって異なります。

第 5 章 AMD uProf GUI の使用開始

5.1 ユーザー インターフェイス

AMD uProf GUI は、パフォーマンス データをプロファイリングして解析するためのビジュアル インターフェイスを提供しています。さまざまなページがあり、各ページにいくつかのサブウィンドウがあります。一番上の水平ナビゲーション バーを使用してページ間を移動できます。ページを選択すると、次に示すように、左端の縦型パネルにサブウィンドウのリストが表示されます。

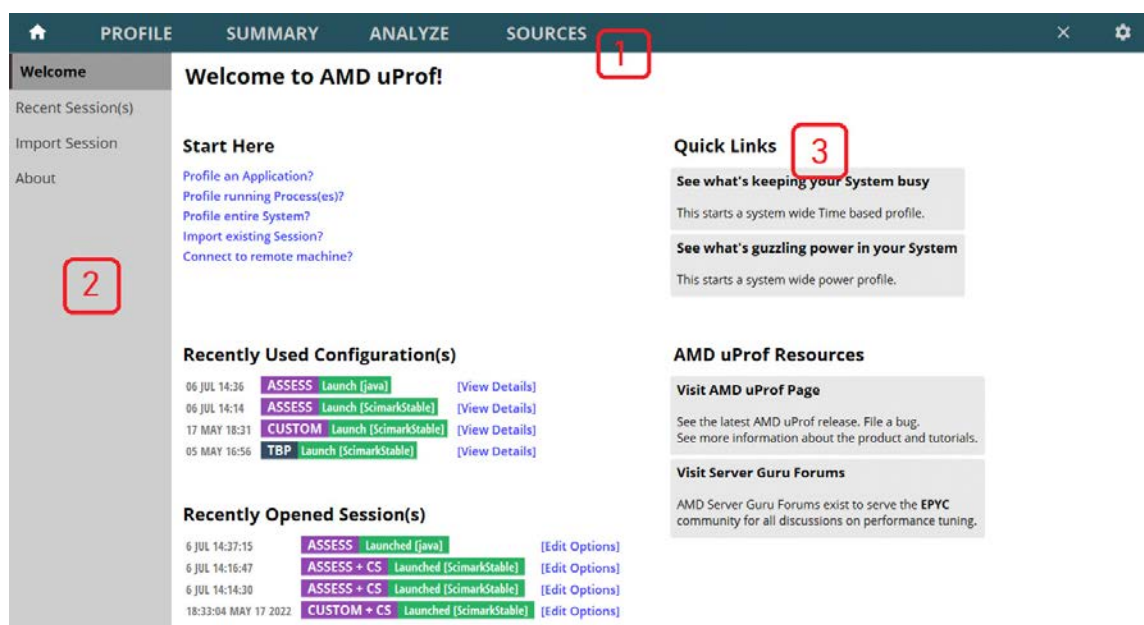


図 3. AMD uProf GUI

1. **[HOME]**、**[PROFILE]**、**[SUMMARY]**、**[ANALYZE]** などの水平バーのメニュー名は、ページと呼ばれます。
2. 各ページのサブウィンドウのリストが、左端の縦型パネルに表示されます。たとえば、**[HOME]** ページには、**[Welcome]**、**[Recent Session(s)]**、**[Import Session]** などの各種ウィンドウがあります。
3. それぞれのウィンドウに、いくつかのセクションがあります。これらのセクションは、プロファイル実行に必要な各種の入力を指定し、プロファイル データを解析用に表示するために使用され、関連セクションへのボタンとリンクも含まれています。**[Welcome]** ウィンドウの **[Quick Links]** セクションにはリンクが 2 つ含まれており、これを使用すると、最小限の設定手順でプロファイル セッションを開始できます。

5.2 GUI の起動

AMDuProf GUI プログラムを起動するには、次の手順に従います。

Windows

`C:\Program Files\AMD\AMDuProf\bin\AMDuProf.exe` を実行するか、デスクトップのショートカットを使用して GUI を起動します。

Linux

`/opt/AMDuProf_X.Y-ZZZ/bin/AMDuProf` バイナリから GUI を起動します。

次のように、[Welcome] 画面が表示されます。

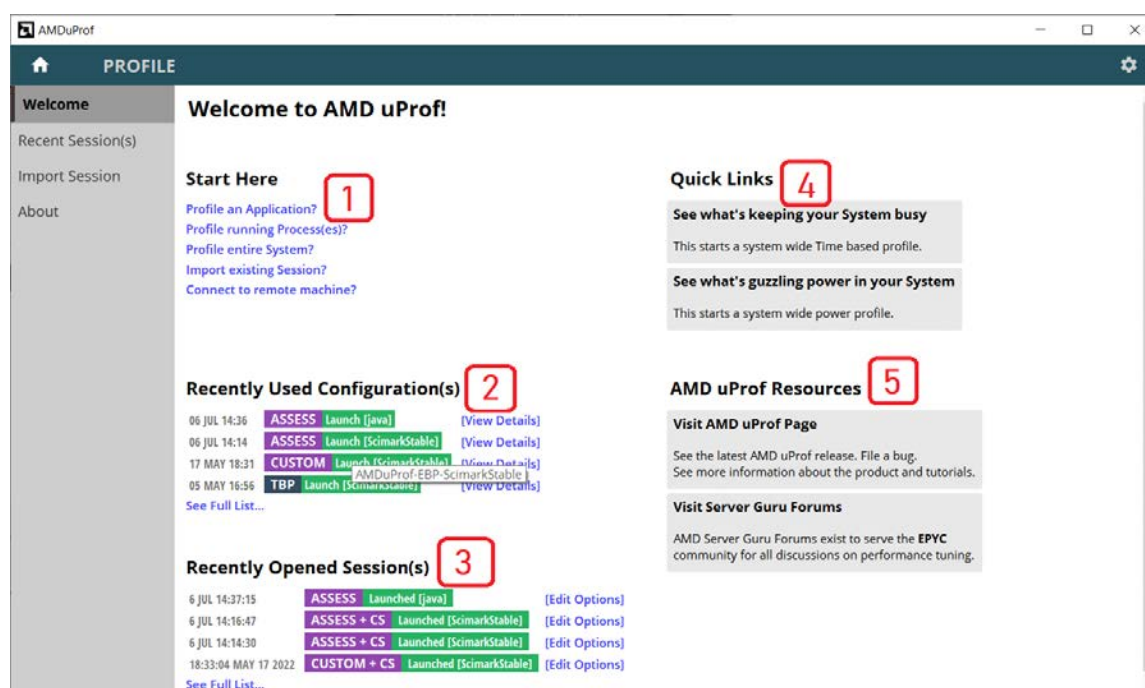


図 4. AMD uProf の [Welcome] 画面

次のセクションが含まれます。

1. **[Start Here]** セクションには、各種プロファイル ターゲットに対してプロファイリングを開始するための クイック リンクがあります。
2. **[Recently Used Configuration(s)]** セクションには、最近使用したプロファイル設定のリストが表示されます。この設定をクリックすると、次のプロファイリングで同じプロファイル設定を再利用できます。
3. **[Recently Opened Session(s)]** セクションには、最近開いたプロファイル セッションのリストが表示されます。いずれかのセッションをクリックして、詳しい分析のために、対応するプロファイル データを読み込むことができます。

4. **[Quick Links]** セクションには 2 つのエントリがあり、これを使用して最小限の設定でプロファイリングを開始できます。
 - a. **[See what's keeping your System busy]** をクリックすると、システム全体に対する時間ベースのプロファイリングが開始されます。停止すると、収集されたデータが表示されます。
 - b. **[See what's guzzling power in your System]** をクリックすると、消費電力および熱関連の各種カウンターを選択し、そのデータのライブ ビューをグラフで表示できます。
5. **[AMD uProf Resources]** セクションには、AMD uProf のリリース ページと AMD サーバー コミュニティ フォーラムへのリンクが表示されており、プロファイリングとパフォーマンス チューニングに関するディスカッションに参加できます。

5.3 プロファイルの設定

収集を実行するには、はじめにプロファイルの設定として、次に示す内容を指定する必要があります。

1. プロファイル ターゲット
2. プロファイル タイプ
 - a. どのプロファイル データを収集するのか (CPU プロファイル、CPU トレース、GPU トレース、消費電力プロファイル)
 - b. 監視イベント - どのような方法でデータを収集するのか
 - c. 追加のプロファイル データ (必要な場合) - コールスタック サンプル、プロファイル スケジューリングなど

以上はプロファイル設定と呼ばれており (43 ページの「[プロファイル設定](#)」)、測定を正しく実行するために使用されるすべての情報が指定されています。

注記: 収集の必要な追加プロファイル データは、選択するプロファイル タイプによって異なります。

5.3.1 プロファイル ターゲットの選択

プロファイリングを開始するには、上部のナビゲーション バーで **[PROFILE]** ページをクリックするか、**[HOME]** ページの **[Welcome]** 画面で **[Profile an Application?]** リンクをクリックします。**[Start Profiling]** 画面が表示されます。次に示すように、**[Start Profiling]** ウィンドウに **[Select Profile Target]** が表示されます。

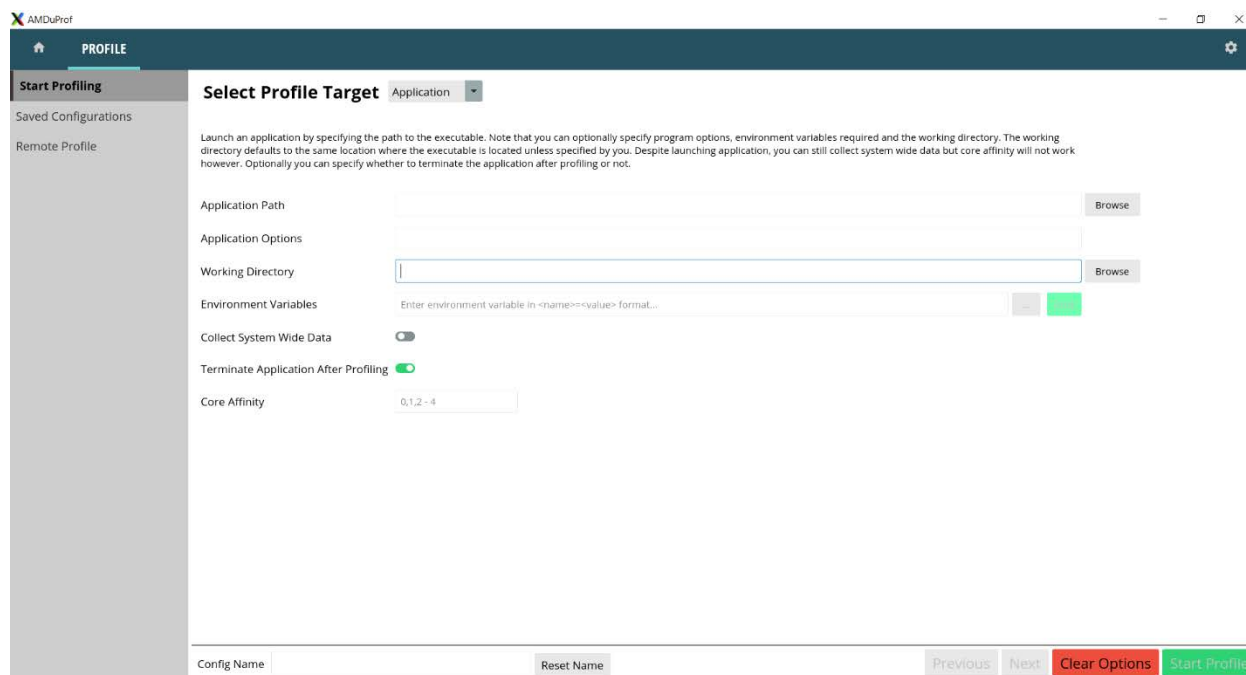


図 5. [Start Profiling] - [Select Profile Target]

[Select Profile Target] ドリルダウンで、次に示すプロファイル ターゲットのいずれかを選択します。

- **[Application]:** アプリケーションを起動してそのアプリケーションをプロファイリングする場合 (またはアプリケーションを起動してシステム全体をプロファイリングする場合) は、このターゲットを選択します。必須のオプションは、実行ファイルへの有効なパスのみです。パスを指定しない限り、実行ファイルへのパスは、デフォルトで作業ディレクトリになります。
- **[System]:** アプリケーションを起動せずに、システム全体のプロファイリングまたは特定のコア セットのプロファイリングを実行する場合は、これを選択します。
- **[Process(es)]:** 既に実行中のアプリケーション/プロセスをプロファイリングする場合は、これを選択します。選択すると、プロセス テーブルが表示されます。このテーブルは更新できます。プロファイリングを開始するには、テーブルからいずれかのプロセスを選択する必要があります。

プロファイル ターゲットを選択して有効なデータを設定すると、**[Next]** ボタンが有効になり、**[Start Profiling]** の次の画面に移動できます。

注記: **[Next]** ボタンが有効になるのは、選択されたオプションがすべて有効な場合のみです。

5.3.2 プロファイル タイプの選択

プロファイル ターゲットを選択して設定したら、[Next] ボタンをクリックします。次のように、[Select Profile Configuration] 画面が表示されます。

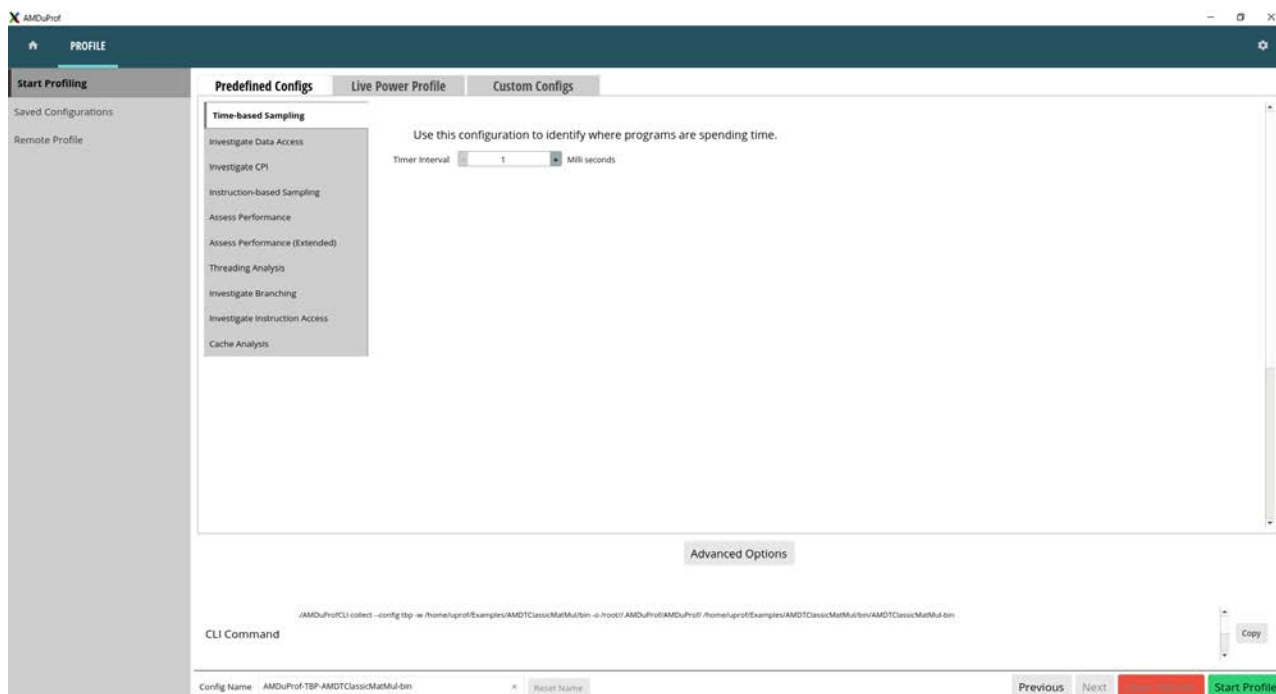


図 6. [Start Profiling] - [Select Profile Configuration]

この画面では、収集するプロファイル データの種類とデータ収集方法を決定します。プロファイル タイプは、実行する予定のパフォーマンス解析に基づいて選択します。上の図で、次のとおりに実行します。

- 次のいずれかのタブを選択します。
 - [Predefined Configs] を構成するのは、すべて事前定義設定であり、[Time-based Profiling]、[Cache Analysis]、[Assess Performance] などが含まれます。
 - [Live Power Profiling] には、リアルタイムの消費電力プロファイリングを実行するオプションが含まれます。
 - [Custom Configs] には、カスタム CPU プロファイル、CPU トレース、GPU トレースを実行するオプションが含まれます。
- プロファイル タイプを選択すると、ウィンドウ内左側の縦型パネルに、選択されたプロファイル タイプに対応するオプションのリストが表示されます。[CPU Profile] タイプの場合、使用可能な事前定義サンプリング設定の一覧が表示されます。
- 修正イベント オプションを使用できるのは、事前定義設定の場合のみです。
- [Advanced Options] ボタンをクリックして、[Advanced Options] 画面に進み、[Call Stack Options]、[Profile Scheduling]、[Sources, Symbols] など、その他のオプションを設定します。

5. 43 ページの「プロファイル設定」の情報は永続的であり、ツールによって名前付き (ここでは *AMDuProf-EBP-ScimarkStable*) で保存されます。この名前を自分で定義しておくで、[PROFILE] → [Saved Configurations] に移動して、後から同じ設定を再利用/選択できます。
6. [Next] および [Previous] ボタンを使用して、[Start Profiling] のさまざまな画面間を移動できます。
このページの下部で CLI コマンドを使用できます。ここには、[Select Profile Configuration] ページで選択された GUI オプションの CLI バージョンが表示されます。

5.3.3 Advanced Options

[Select Profile Type] 画面で [Advanced Options] ボタンをクリックします。次のように、[Advanced Options] 画面が表示されます。

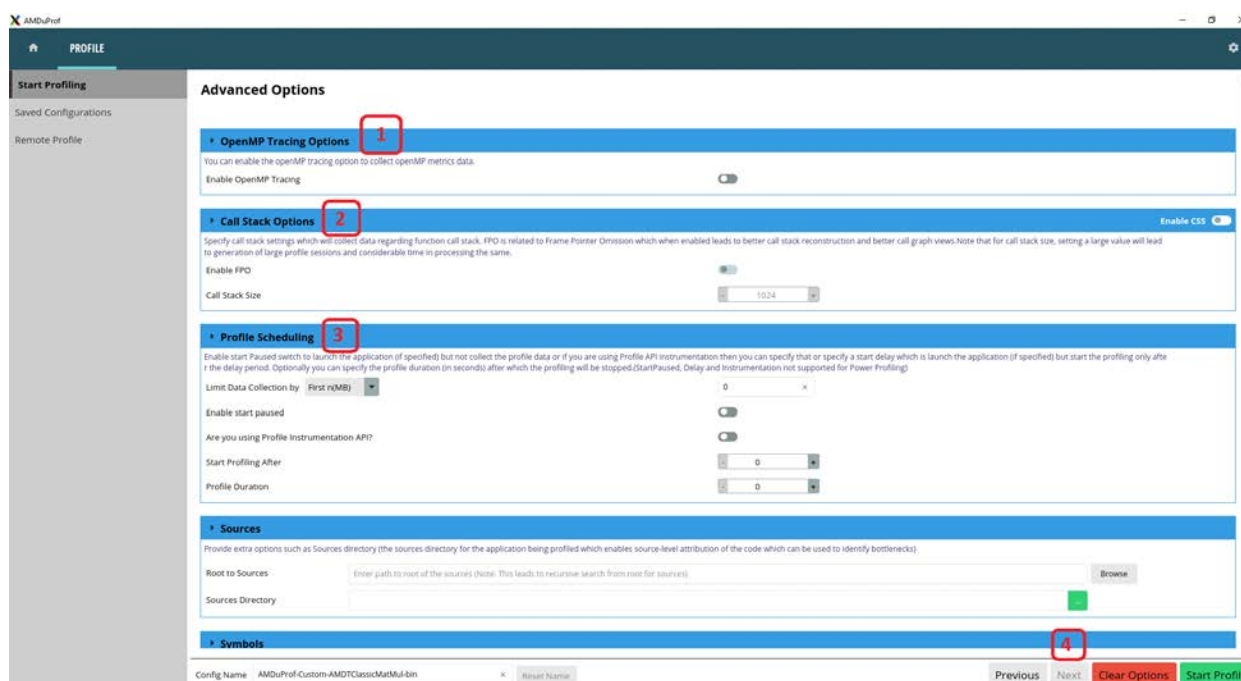


図 7. [Start Profiling] - [Advanced Options] 1

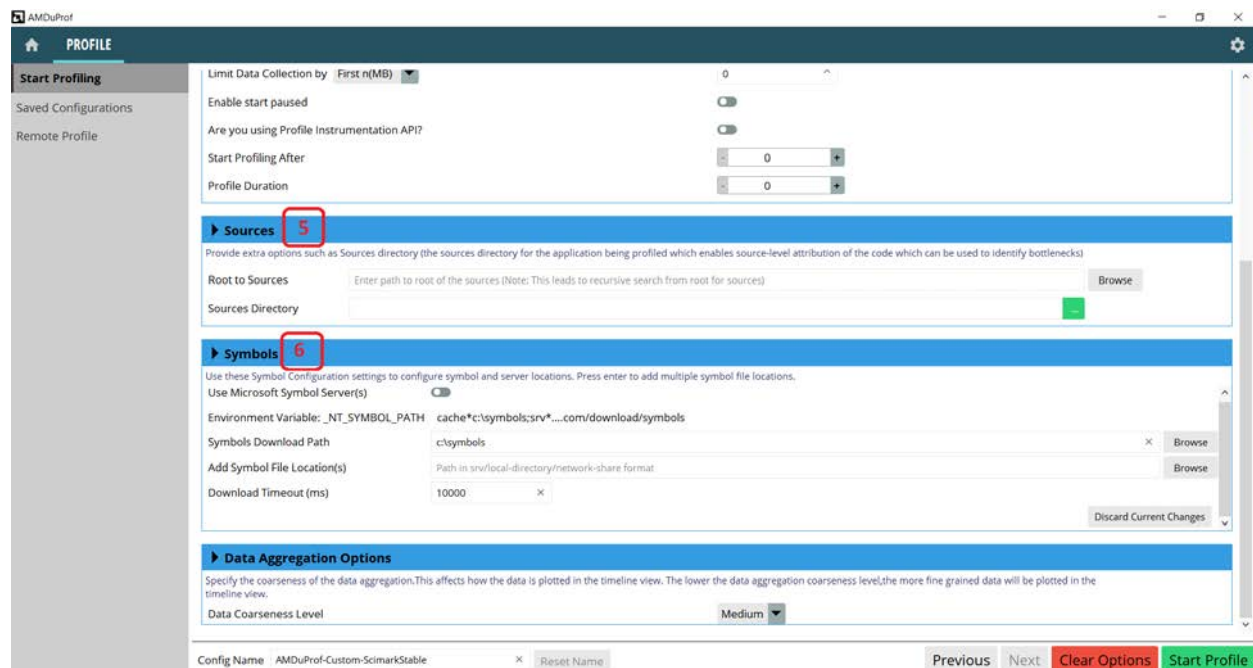


図 8. [Start Profiling] - [Advanced Options] 2

[Advanced Options] 画面では、次のオプションを設定できます。

1. [Enable Thread Concurrency] では、プロファイル データを収集して、Windows でスレッド同時実行性グラフを表示できます。
2. [Call Stack Options] では、コールスタック サンプルデータの収集を有効にできます。このプロファイルデータを使用して、[Top-Down Callstack]、[Flame Graph]、[Call Graph] ビューを表示できます。
3. [Profile Scheduling] では、プロファイル データの収集をスケジューリングできます。
4. [Next] および [Previous] ボタンを使用して、[Start Profiling] のさまざまなセクション間を移動できます。
5. [Sources] の行編集ボックスで、プロファイル対象アプリケーションのソース ファイルの場所を示すパスを指定します。
6. [Symbols] では、シンボル サーバーを指定し (Windows のみ)、プロファイル対象アプリケーションのシンボル ファイルの場所を示すパスを指定します。

また、[Download timeout] に、サーバーからのシンボル ファイルダウンロードのタイムアウトを指定できます。

5.3.4 プロファイリングの開始

すべてのオプションを正しく設定したら、**[Start Profile]** ボタンをクリックしてプロファイリングを開始し、プロファイル データを収集します。プロファイルの初期化の後で、次の画面が表示されます。

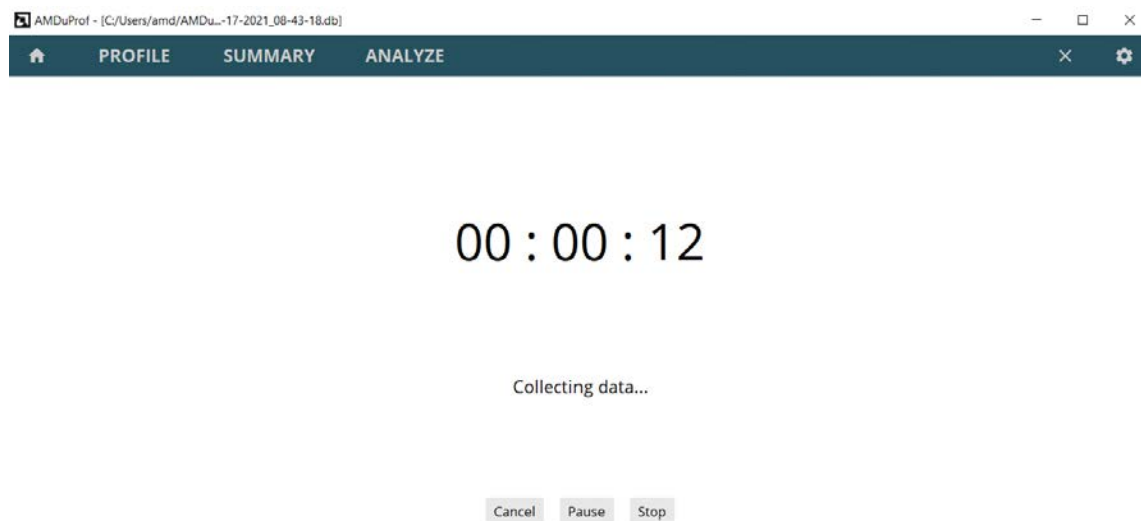


図 9. プロファイル データの収集

1. データ収集中の経過時間が表示されます。
2. プロファイリングの進行中に、次の操作を実行できます。
 - **[Stop]** ボタンをクリックすると、プロファイリングが停止します。
 - **[Cancel]** ボタンをクリックすると、プロファイリングがキャンセルされます。キャンセルすると、**[PROFILE]** の **[Select Profile Target]** 画面に戻ります。
 - **[Pause]** ボタンをクリックすると、プロファイリングが一時停止されます。プロファイル データが収集されなくなり、**[Resume]** ボタンをクリックするとプロファイリングを継続できます。

5.4 変換の進行状況

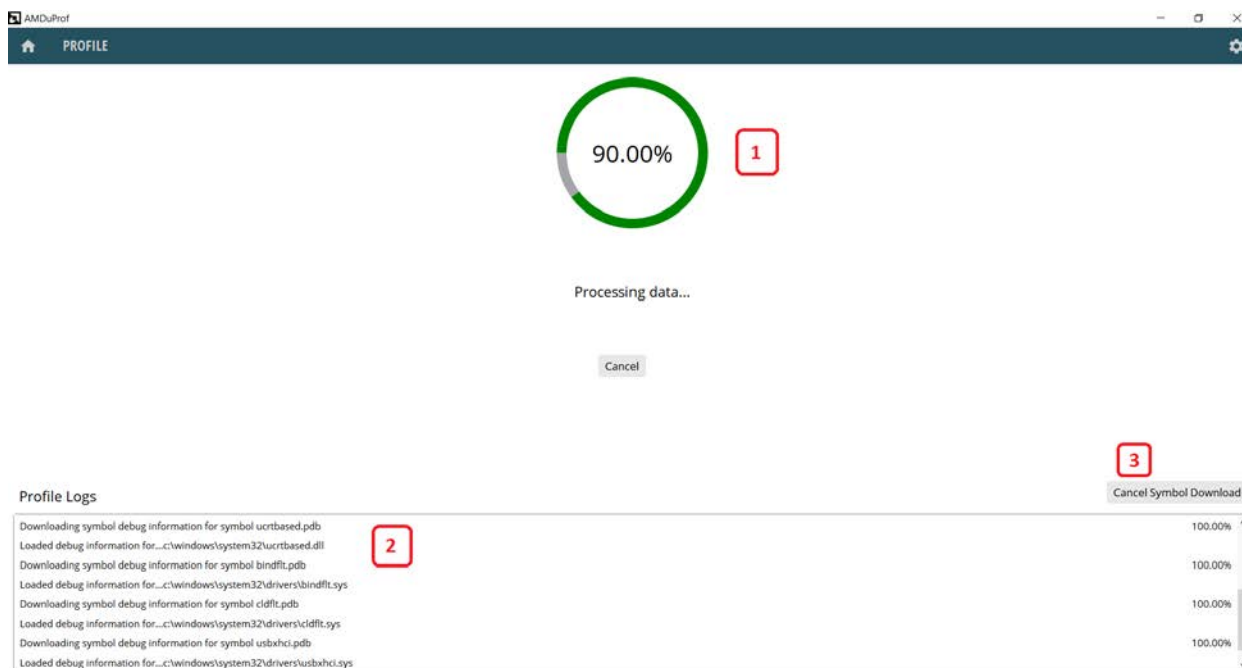


図 10. 変換の進行状況

この画面には次の情報が表示されます。

1. 完了した変換の割合。
2. [Profile Logs] には、現在読み込まれているシンボルと、対応するダウンロードの割合が表示されます。
3. [Cancel Symbol Download] ボタンをクリックすると、シンボル設定ページで指定されたサーバーからのシンボルのダウンロードが停止します。

注記: このオプションは Windows のみで使用できます。

5.5 プロファイル データの解析

プロファイリングが停止すると、収集された生のプロファイル データが自動的に処理されます。ユーザーは、次の UI セクションからプロファイル データを解析して潜在的なパフォーマンス ボトルネックを特定できます。

- [SUMMARY] ページでは、プロファイル セッションのホットスポットの概要を確認できます。
- [ANALYZE] ページでは、プロファイル データをさまざまな粒度で調査できます。
- [SOURCES] ページでは、データをソース行レベルとアセンブリ レベルで調査できます。
- [MEMORY] ページでは、潜在的なキャッシュの偽共有がないかどうかについて、キャッシュ ライン データを調査できます。
- [HPC] ページでは、潜在的なロード アンバランスの問題がないかどうかについて、OpenMP トレース データを調査できます。

- **[TIMECHART]** ページでは、MPI API トレース、OS イベント トレース、情報をタイムライン チャートとして視覚化できます。

使用可能なセクションは、プロファイル タイプによって異なります。**CPU プロファイル**の場合、**[SUMMARY]**、**[ANALYZE]**、**[MEMORY]**、**[HPC]**、**[TIMECHART]**、**[SOURCES]** ページを使用して、データを解析できます。

5.5.1 パフォーマンス ホットスポットの概要

変換が完了すると、**[SUMMARY]** ページにプロファイル データが読み込まれ、**[Hot Spots]** 画面が表示されます。**[SUMMARY]** ページでは、**[Hot Spots]** と **[Session Information]** といった画面を介して、プロファイル セッションのホットスポットの概要が提供されます。

[Hot Spots] 画面には、関数、モジュール、プロセス、スレッドのホットスポットが表示されます。プロセスとスレッドが表示されるのは、複数ある場合のみです。

次の図に、**[Hot Spots]** 画面を示します。

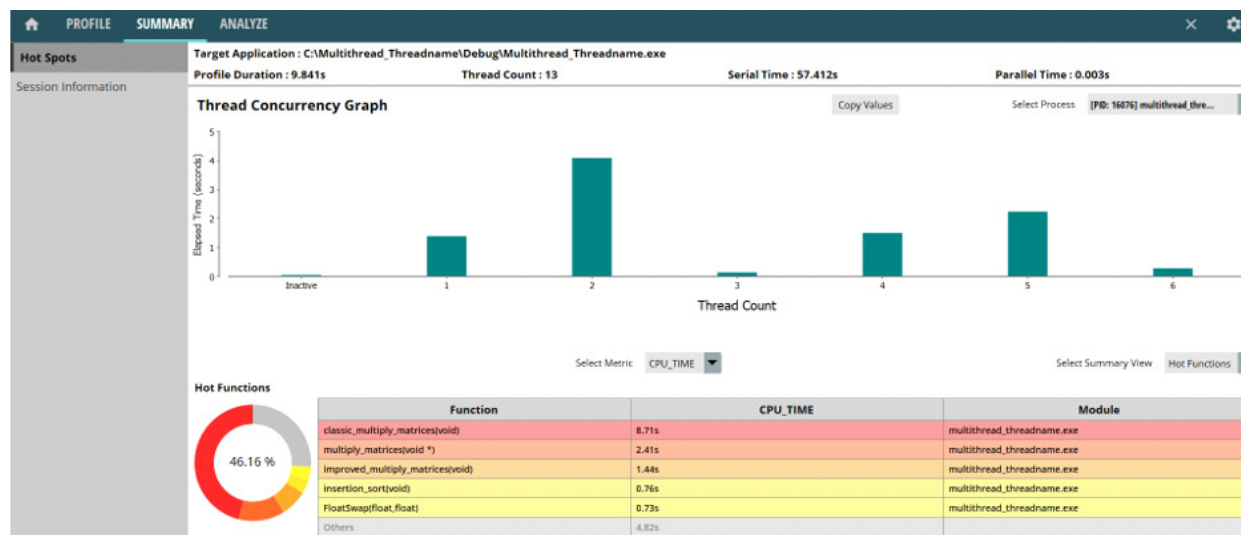


図 11. [Summary] - [Hot Spots]

上の **[Hot Spots]** 画面には次の内容が表示されます。

1. 選択されたイベントに対する最もホットな 5 つの関数、プロセス、モジュール、スレッドが表示されます。
2. **[Hot Functions]** 円グラフはインタラクティブな性質を持っています。いずれかの部分をクリックすると、対応する関数のソースが、**[SOURCES]** ページの独立したタブに表示されます。
3. ホットスポットはイベントごとに表示され、監視対象イベントを右上のドロップダウンから選択できます。別のイベントに変更すると、対応するホットスポット データが更新されます。

4. **[Select Summary View]** ドロップダウンで、次のいずれかを選択します。

- **[Hot Threads]**
- **[Hot Processes]**
- **[Hot Functions]**
- **[Hot Modules]**

選択に応じて、一度に 1 つの円グラフが表示されます。

Summary Overview

選択した内容に基づいて、**[Summary Overview]** 画面は次のようになります。

表 22. Summary Overview

収集されるデータ	表示されるテーブル	説明	タイミングの詳細
OS トレース	Schedule Summary	スレッドあたりの実行/待機時間のサマリ (%)。	• プロファイル時間 • 並列時間 • シリアル時間 • 待機時間 • スリープ時間
	Wait Object Summary	いくつかの種類の同期オブジェクト (ロック、ミューテックス、条件変数など) に関連するオペレーションで消費された時間。	
	Wait Function Summary	いくつかの種類の pthread ブロッキング関数 (pthread_join など) で消費された時間。	
	Syscall Summary	syscall で消費された時間	
GPU トレース	GPU Kernel Summary	エンキューされたデバイスでの実行に消費された GPU カーネルあたりの時間。	• プロファイル時間
	Data Transfer Summary	GPU データのコピー オペレーションで消費された時間。	
MPI トレース	MPI P2P API Summary	すべてのプロファイル ランクを通じて各種 MPI P2P API で消費された時間。	• プロファイル時間 • 並列時間 • シリアル時間 • MPI Time
	MPI Collective API Summary	すべてのプロファイル ランクを通じて、各種 MPI 集合型通信 API で消費された時間。	
CPU プロファイル	Hot Functions	CPU プロファイルに基づいて最もホットな関数。	• プロファイル時間 • 並列時間 • シリアル時間
	Hot Modules	CPU プロファイルに基づいて最もホットなモジュール。	
	Hot Threads	CPU プロファイルに基づいて最もホットなスレッド。	
	Hot Processes	CPU プロファイルに基づいて最もホットなプロセス。	

OS Trace

[OS Trace] 画面は次のようになります。

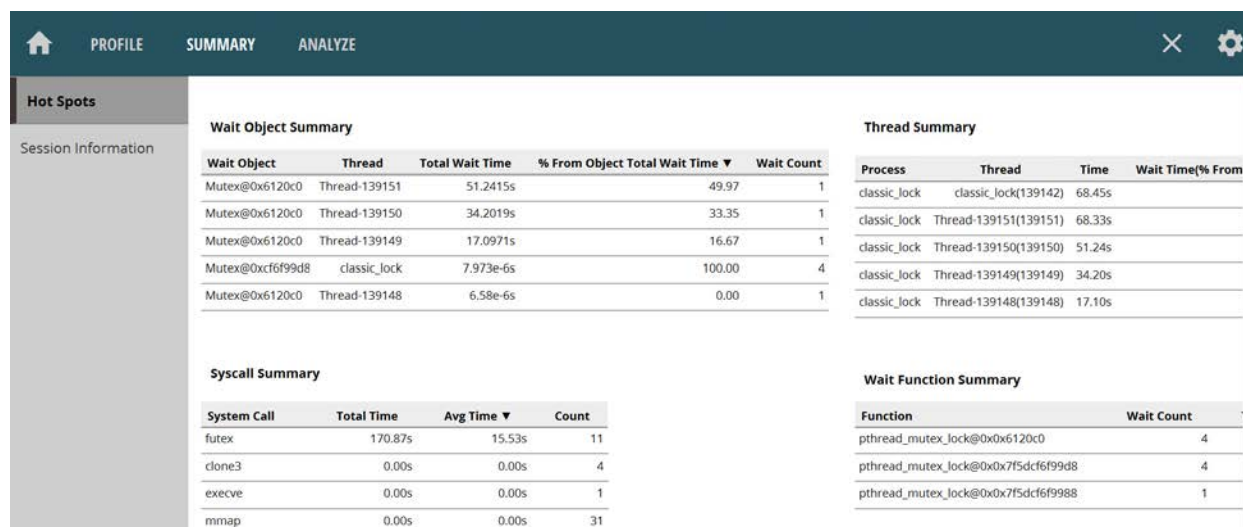


図 12. [OS Trace]

GPU Trace

[GPU Trace] 画面は次のようになります。

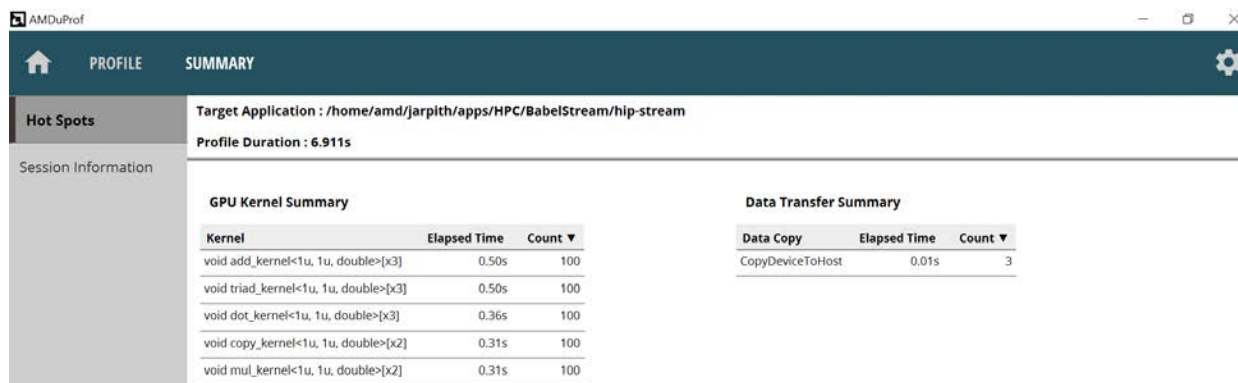
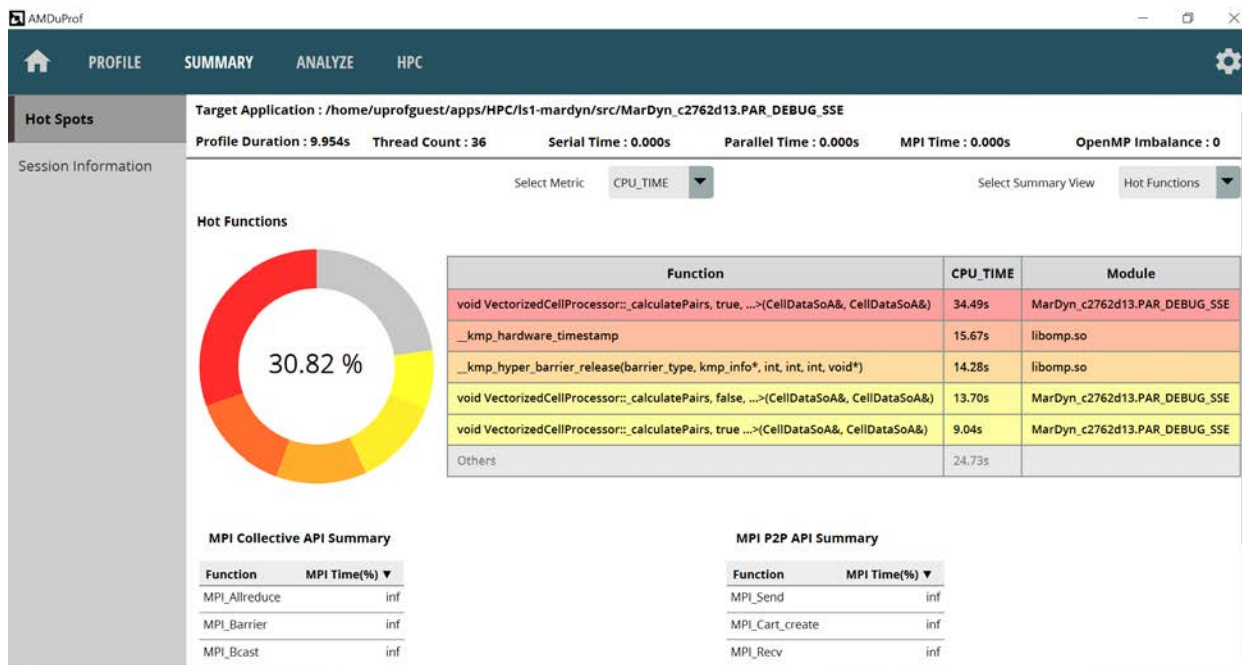


図 13. [GPU Trace]

MPI Trace

[MPI Trace] 画面は次のようになります。



CPU Profile

[CPU Profile] 画面は図 11 のようになります。

5.5.2 スレッド同時実行性グラフ

[ANALYZE] → [Thread Concurrency] をクリックすると、次のグラフが表示され、プロファイル対象アプリケーションのスレッド同時実行性を解析できます。

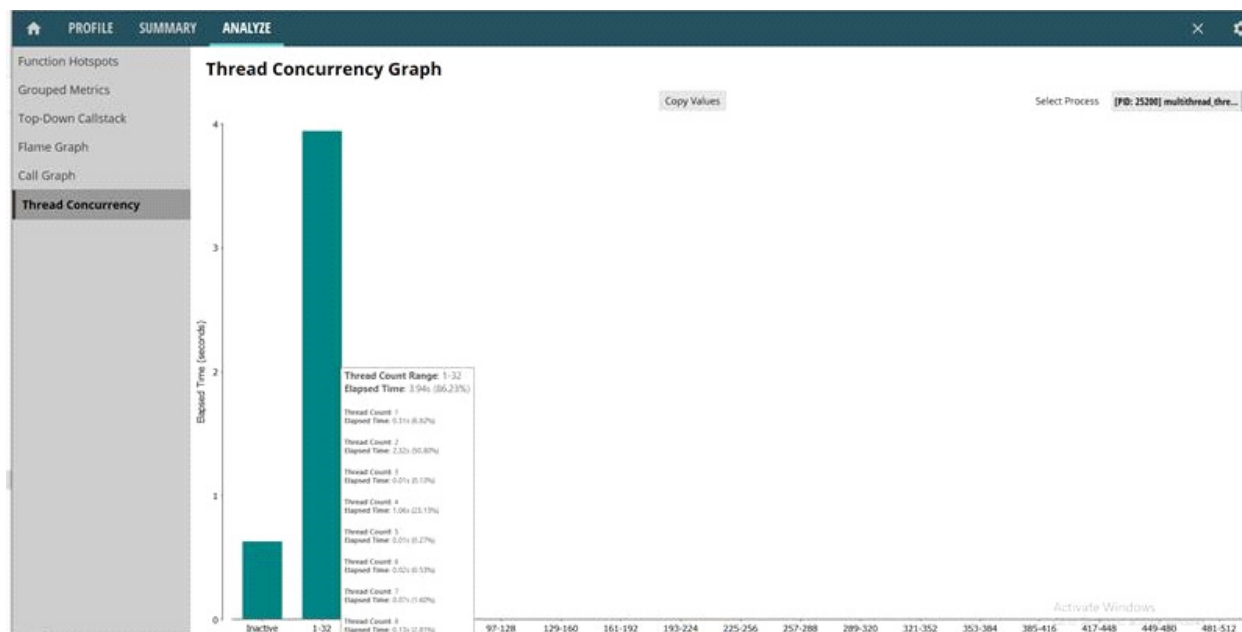


図 14. [Summary] - [Thread Concurrency Graph]

スレッド同時実行性グラフには、同時に実行されていた特定数のスレッドの持続時間 (秒) が表示されます。このグラフでは、バケット化アプローチが使用されています。各コアの経過時間を表示する代わりに、バケット サイズに基づいて重み付けされた平均が算出されます。バケット サイズは、コアと、使用可能なピクセル数に基づいて決定されます。これは、横方向のスクロールを避けるための処理です。

5.5.3 関数ホットスポット

上部の水平ナビゲーションバーで **[ANALYZE]** をクリックして、**[Function Hotspots]** 画面に移動します。この画面には、次に示すように、プロファイルされるすべてのプロセスおよびロード モジュールを通じてホットな関数が表示されます。

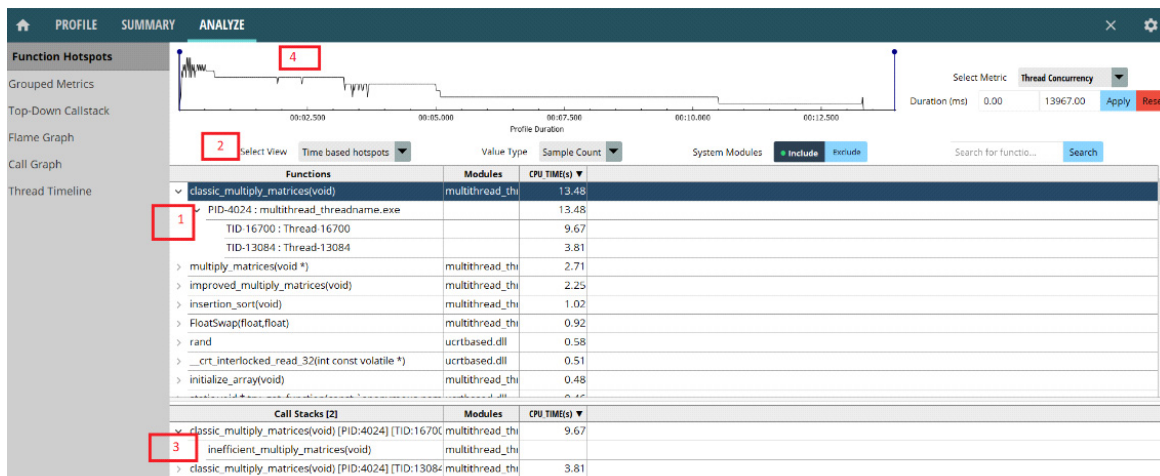


図 15. [ANALYZE] - [Function Hotspots]

[Function Hotspots] 画面には、次の内容が含まれます。

- [Function Hotspots]** ビューでエントリを開くと、プロセスおよびスレッドに関するデータ分類を表示できます。**[Functions]** テーブルにホットな関数のリストが表示されます。IP サンプルが集計され、関数レベルの粒度で原因が特定されます。このテーブルでは、次の操作を実行できます。
 - 関数エントリをダブルクリックすると、対応するその関数の **[SOURCE]** ビューに移動できます。
 - 右クリックして次のオプションから選択できます。
 - [Copy selected row(s)]** を選択すると、ハイライトした行がクリップボードにコピーされます。
 - [Copy all rows]** を選択すると、すべての行がクリップボードにコピーされます。
- [Filters and Options]** パネルでは、次に示すように、データをフィルターしてプロファイリングできます。
 - [Select View]** ドロップダウンをクリックすると、表示されるカウンターを操作できます。関連するカウンターとそれらから算出されたメトリクスは、事前定義ビューとしてグループ化されています。
 - [Value Type]** ドロップダウンを使用すると、次のようにカウンターの値を表示できます。
 - [Sample Count]** は、特定の関数に起因するサンプルの数です。
 - [Event Count]** は、サンプル数とサンプル間隔を掛けたものです。
 - [Percentage]** は、特定の関数に対して収集されたサンプルの割合です。
 - [System Modules]** オプションでは、**[Exclude]** または **[Include]** を使用して、システム モジュールに起因するプロファイル データを除外するか、含めるかを指定できます。
- コールスタックが有効になっている場合、選択された関数に対する固有のホット コールパスが **[Functions]** 列に表示されます。

4. **[Event Timeline]** は、特定の期間中に集計されたサンプル値の数を示す折れ線グラフです。これは、プロファイル領域内のホット関数を特定するために使用できます。**[Select Metric]** ドロップダウンからイベントを選択し、そのイベント タイムラインをプロットできます。

特定のプロファイルに対してすべてのエントリが読み込まれるわけではありません。デフォルトを超える数のエントリを読み込むには、右側の縦型スクロールバーをクリックします。エントリを開くと、プロセスおよびスレッドに関するデータ分類を表示できます。

5.5.4 プロセスと関数

[ANALYZE] → **[Grouped Metrics]** をクリックすると、プロセス、ロード モジュール、スレッド、関数といった各種のプログラム ユニット粒度でプロファイル データ テーブルを表示できます。この画面には、2 つの異なる形式でデータが表示されます。

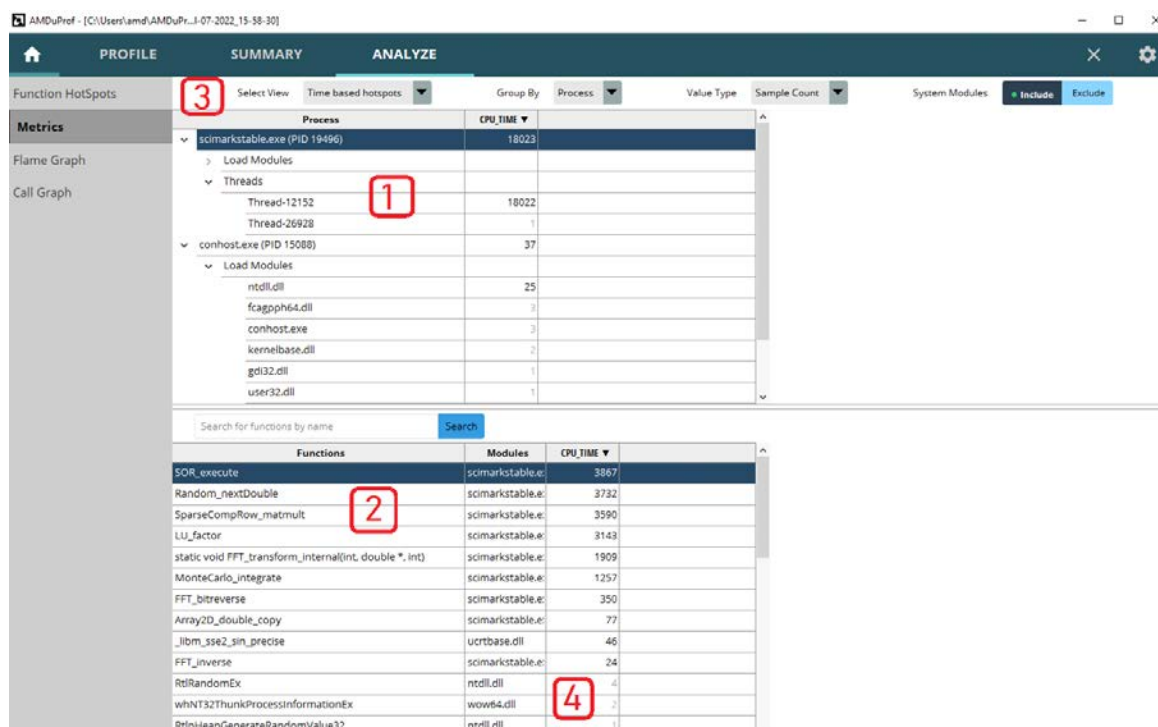


図 16. [Analyze] - [Metrics]

上に示した図の構成は次のとおりです。

1. 上側のツリーは、**プロセス**別にグループ化されたサンプルを表しています。ツリーを開くと、それぞれの親 (プロセス) の子エントリを表示できます。**ロード モジュール**と**スレッド**は、選択したプロセス エントリの子エントリになります。

右クリックすると、次のオプションが表示されます。

- **[Expand All Entries]** を選択すると、すべてのプロセスのモジュールおよびスレッドのリストが表示されます。
- **[Collapse All Entries]** を選択すると、最上位のエントリのリストのみが表示されます。
- **[Copy selected row(s)]** を選択すると、ハイライトした行がクリップボードにコピーされます。
- **[Copy all rows]** を選択すると、すべての行がクリップボードにコピーされます。

2. 下側の **[Functions]** テーブルには、対応する関数に起因するサンプルが含まれています。関数のエントリは、上側のツリーで選択された項目によって異なります。より詳しいデータを表示するには、上側のツリーで子エントリを選択すると、下側のツリーで対応する関数データが更新されます。次の操作を実行できます。

- 関数エントリをダブルクリックすると、対応する **[SOURCE]** ビューに移動できます。
- 右クリックして次のオプションから選択できます。
 - **[Copy selected row(s)]** を選択すると、ハイライトした行がクリップボードにコピーされます。
 - **[Copy all rows]** を選択すると、すべての行がクリップボードにコピーされます。

3. **[Filters and Options]** パネルを使用すると、各種コントロールによって表示されるプロファイル データをフィルターできます。

- **[Select View]** では、表示されるカウンターを制御できます。関連するカウンターとそれらから算出されたメトリクスは、事前定義ビューとしてグループ化されています。**[Select View]** ドロップダウンからビューを選択できます。
- **[Group By]** ドロップダウンを使用すると、プロセス、モジュール、スレッド別にデータをグループ化できます。デフォルトでは、サンプル データはプロセス別にグループ化されています。
- **[Value Type]** ドロップダウンをクリックして、次のようにカウンターの値を表示できます。
 - **[Sample Count]** は、特定の関数に起因するサンプルの数です。
 - **[Event Count]** は、サンプル数とサンプル間隔を掛けたものです。
 - **[Percentage]** は、特定の関数に対して収集されたサンプルの割合です。
- **[System Modules]** オプションでは、**[Exclude]** または **[Include]** を使用して、システム モジュールに起因するプロファイル データを除外するか、含めるかを指定できます。

4. 信頼性レベル — プログラム ユニットに対して収集されたサンプル数が少ないため、信頼性をもって計算できないメトリクスは、グレーで表示されます。

特定のプロファイルに対してすべてのエントリが読み込まれるわけではありません。デフォルトを超える数のエントリを読み込むには、右側の縦型スクロール バーをクリックします。

5.5.5 ソースとアセンブリ

[Metrics] 画面の [Functions] テーブルで任意のエントリをダブルクリックすると、[SOURCES] ページに対応する関数のソース タブが読み込まれます。GUI によって該当関数のソース ファイルのパスが見つかる場合、ファイルのオープンが試行されます。これに失敗すると、ファイルの場所を指定するよう要求するプロンプトが表示されます。

次に、ソースとアセンブリを表示した画面を示します。

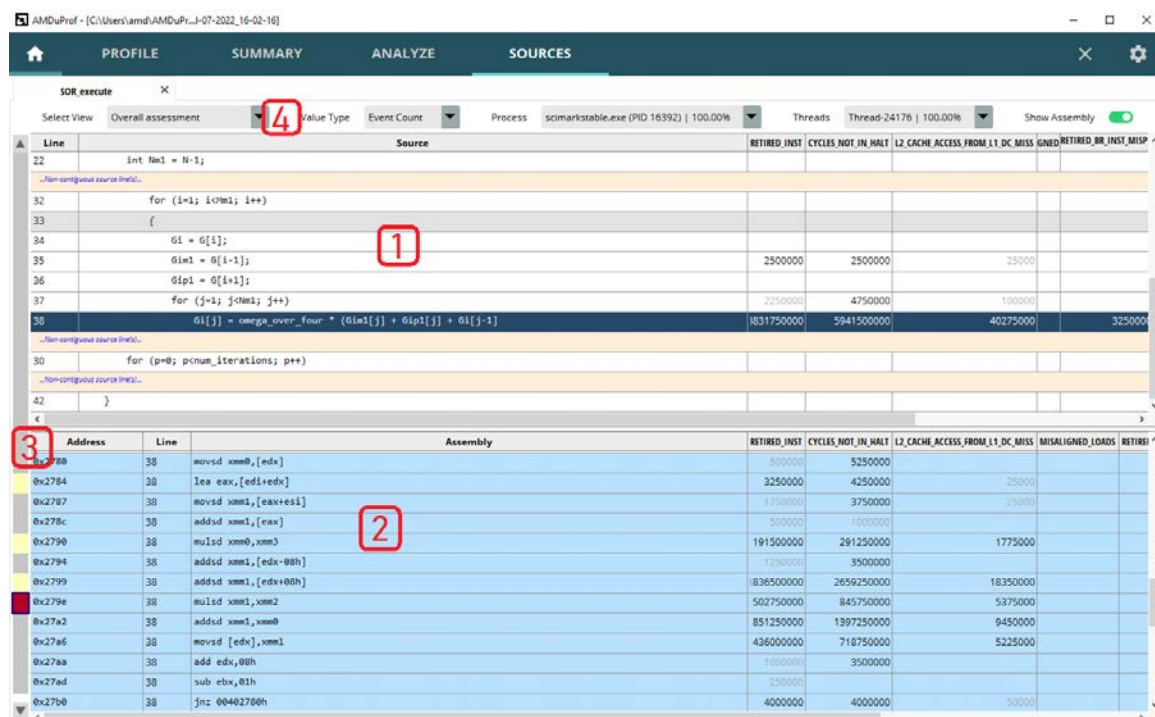


図 17. [SOURCES] - ソースとアセンブリ

[SOURCES] 画面には次のセクションが表示されます。

1. 選択された関数のソース行がリスト表示され、対応するメトリクスが各ソース行の異なる列内に読み込まれます。ソース行が実行されたときにサンプルが収集されていない場合、メトリクスの列は空白になります。
2. ハイライトされたソース行に対応するアセンブリ命令。ツリーには、各アセンブリ命令のオフセットがメトリクスと共に表示されます。
3. ヒートマップ — ソース レベルでのホットスポットの概要。

4. フィルター パネルでは、次のオプションを指定することでプロファイル データをフィルターできます。
- **[Select View]** では、表示されるカウンターを制御できます。関連するカウンターとそれらから算出されたメトリクスは、事前定義ビューとしてグループ化されています。**[Select View]** ドロップダウンからビューを選択できます。
 - **[Process]** ドロップダウンには、選択された関数が実行されて、サンプルが取得されたすべてのプロセスのリストが表示されます。
 - **[Threads]** ドロップダウンには、選択されたこの関数が実行されて、サンプルが取得されたすべてのスレッドのリストが表示されます。
 - **[Value Type]** ドロップダウンを使用すると、次のようにカウンターの値を表示できます。
 - **[Sample Count]** は、特定の関数に起因するサンプルの数です。
 - **[Event Count]** は、サンプル数とサンプル間隔を掛けたものです。
 - **[Percentage]** は、特定の関数に対して収集されたサンプルの割合です。
 - **[Show Assembly]** ボタンを使用すると、ビュー下部のアセンブリ命令テーブルを表示するかどうかを指定できます。

マルチスレッド アプリケーションまたはマルチプロセス アプリケーションで、特定の関数が複数のスレッドまたはプロセスから実行された場合、それぞれのスレッドまたはプロセスが、**フィルター パネル**の **[Process]** および **[Threads]** ドロップダウンに表示されます。これらのドロップダウンを変更すると、選択に応じたプロファイル データが更新されます。デフォルトでは、選択された関数のプロファイル データが、すべてのプロセスとすべてのスレッドで集計されて表示されます。

注記: ソース ファイルが見つからないか、開かない場合、ディスアセンブルのみが表示されます。

5.5.6 トップダウン コールスタック

[Top-down Callstack] ビューは、アプリケーションのコール シーケンス フローを調べて、関数とその呼び出された側で消費された時間を解析するために使用できます。

[ANALYZE] → **[Top-down Callstack]** をクリックすると、次のように表示されます。

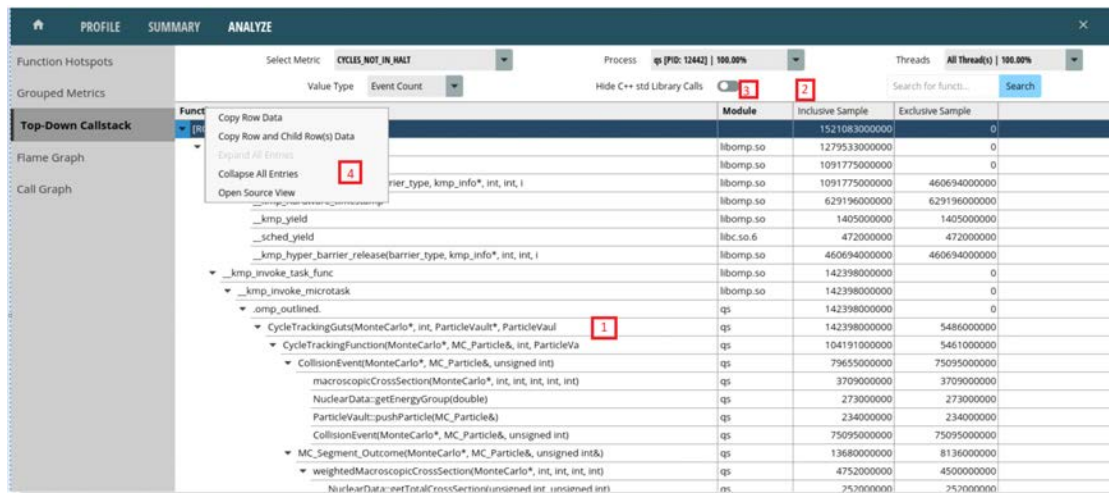


図 18. [Top-down Callstack]

1. 関数は、算入サンプルの並べ替え値に従い、エントリへの親子関係に基づいて表示されます。
2. 特定の関数とその下位関数に対する算入サンプルの値。
3. **[Hide C++ std Library Calls]** オプションが有効になるのは、C++ ライブラリの呼び出しが実行された場合のみです。このオプションを選択すると、C++ ライブラリの呼び出しがリストから除外され、その他の子エントリが表示されます。
4. コンテキスト メニューでエントリの非展開を選択すると、開かれていたエントリがすべて閉じられます。エントリの展開を選択すると子エントリが表示され、**[Open Source View]** オプションを選択すると、対応するソース ビューが開きます。

5.5.7 フレーム グラフ

フレーム グラフは、サンプリングされたコールスタック トレースを視覚化したもので、最もホットなコード実行パスをすばやく特定できます。**[ANALYZE]** → **[Flame Graph]** をクリックすると、次のように表示されます。

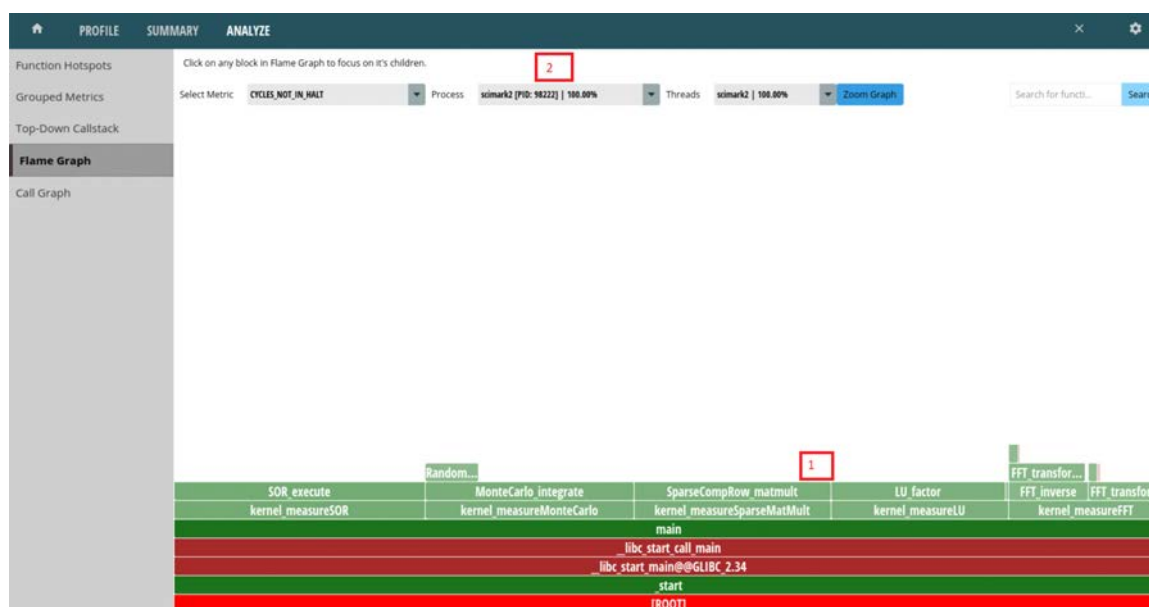


図 19. **[ANALYZE]** - **[Flame Graph]**

[Flame Graph] 画面の構成は次のとおりです。

1. フレーム グラフの X 軸はコールスタック プロファイルを示し、Y 軸はスタックの深さを示します。時間経過に基づくプロットではありません。それぞれのセルがスタック フレームを表し、コールスタック サンプルにフレームが含まれる回数が多いほど、そのセルの幅が広がります。この画面には次のオプションがあります。
 - モジュール別にセルを色分けできます。
 - いずれかのセルをクリックすると、そのセルおよび子セルのみにズームできます。**[Reset Zoom]** ボタンを使用すると、グラフ全体を視覚化できます。

- セルを右クリックすると、次のコンテキスト オプションが表示されます。
 - [Copy Function Data] は、関数の名前とメトリクスをクリップボードにコピーします。
 - [Open Source View] は、その関数のソース タブに移動します。
- いずれかのセルにカーソルを合わせるとツール ヒントが表示され、その関数の算入サンプル数と除外サンプル数が示されます。

2. この画面の上部には次のオプションがあります。

- [Zoom Graph] ボタンをクリックして、ズーム エクスペリエンスを改善できます。
- 検索ボックスに関数名を入力すると、適合する関連関数がすべて表示されます。目的の関数を選択すると、その関数に対応するセルがフレーム グラフ内でハイライトされます。
- [Process] ドロップダウンには、コールスタック サンプルが収集されたプロセスがすべて表示されます。プロセスを変更すると、その特定プロセスのフレーム グラフがプロットされます。
- マルチスレッド アプリケーションの場合、フレーム グラフにはデフォルトで、すべてのスレッドを累積したデータがプロットされます。
- [Threads] ドロップダウンには、コールスタック サンプルが収集されたスレッドがすべて表示されます。スレッドを変更すると、そのスレッドのフレーム グラフがプロットされます。
- [Select Metric] ドロップダウンには、コールスタック サンプルが収集されたメトリクスがすべて表示されます。メトリクスを変更すると、その特定メトリクスのフレーム グラフがプロットされます。

5.5.8 コールグラフ

[ANALYZE] → [Call Graph] をクリックすると、コールグラフ画面に移動します。このグラフは、コールスタック サンプルを使用して構成されており、ホットなコールパスを解析するためのバタフライビューを提供します。

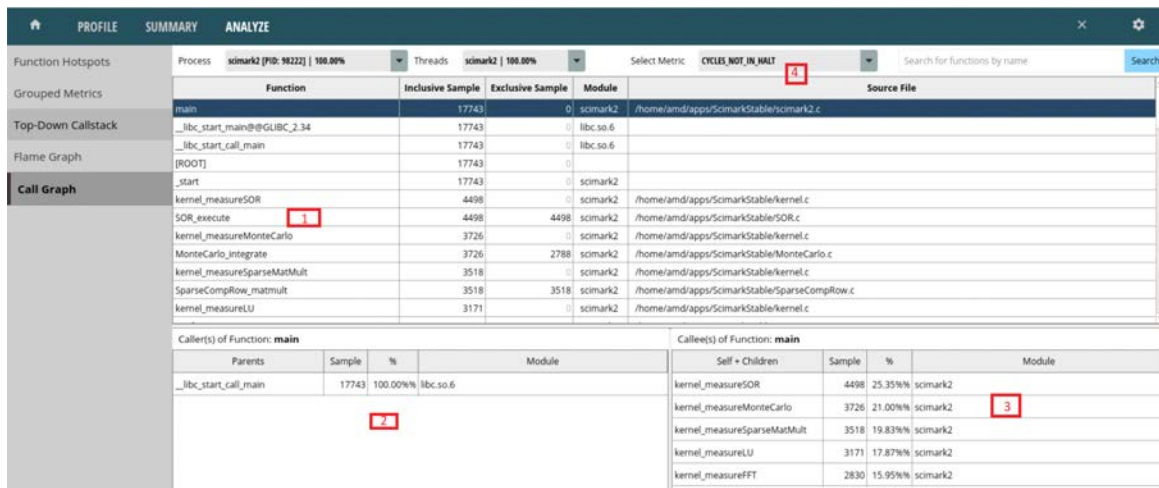


図 20. [ANALYZE] - [Call Graph]

- 関数テーブルには、すべての関数と算入サンプルおよび除外サンプルが表示されます。
いずれかの関数をクリックすると、その呼び出し元関数と呼び出された関数がバタフライビューに表示されます。
- 関数テーブルで選択された関数の親がすべて表示されます。
- 関数テーブルで選択された関数の子がすべて表示されます。
- オプションは次のとおりです。
 - [Process]** ドロップダウンには、コールスタック サンプルが収集されたプロセスがすべて表示されます。プロセスを変更すると、その特定プロセスのコールグラフが表示されます。
 - マルチスレッド アプリケーションの場合、コールグラフにはデフォルトで、すべてのスレッドを累積したデータがプロットされます。
 - [Threads]** ドロップダウンには、コールスタック サンプルが収集されたスレッドがすべて表示されます。スレッドを変更すると、そのスレッドのコールグラフがプロットされます。
 - [Select Metric]** ドロップダウンには、コールスタック サンプルが収集されたメトリクスが表示されます。カウンターを変更すると、その特定カウンターのコールグラフが表示されます。

5.5.9 IMIX ビュー

IMIX ビューは、収集されたサンプルを命令別のサマリーで表示します。このビューが表示されるのは、IBS プロファイリングのみです。**[ANALYZE]** → **[IMIX]** をクリックすると、IMIX ビューに移動します。

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

図 21. IMIX ビュー

- IMIX テーブルには、すべての命令とサンプル数に加えて、選択されたオプションのサンプルの割合が表示されます。
- オプションは次のとおりです。
 - [Select Metric]** ドロップダウンには、サンプルが収集されたメトリクスがすべて表示されます。メトリクスを変更すると、そのメトリクスの IMIX 情報が表示されます。
 - [Module]** ドロップダウンには、そのサンプルが収集されたバイナリがすべて表示されます。モジュールを変更すると、そのモジュールの IMIX 情報が表示されます。

- **[Functions]** ドロップダウンには、サンプルが収集された関数がすべて表示されます。関数を変更すると、そのスレッドの IMIX 情報が表示されます。デフォルトでは、すべての関数の IMIX 情報が表示されます。

5.6 プロファイル データベースのインポート

CLI を使用して生成されたプロファイル データベースを解析するには、**[HOME]** → **[Import Session]** をクリックして **[Import Profile Session]** に移動します。次の画面が表示されます。

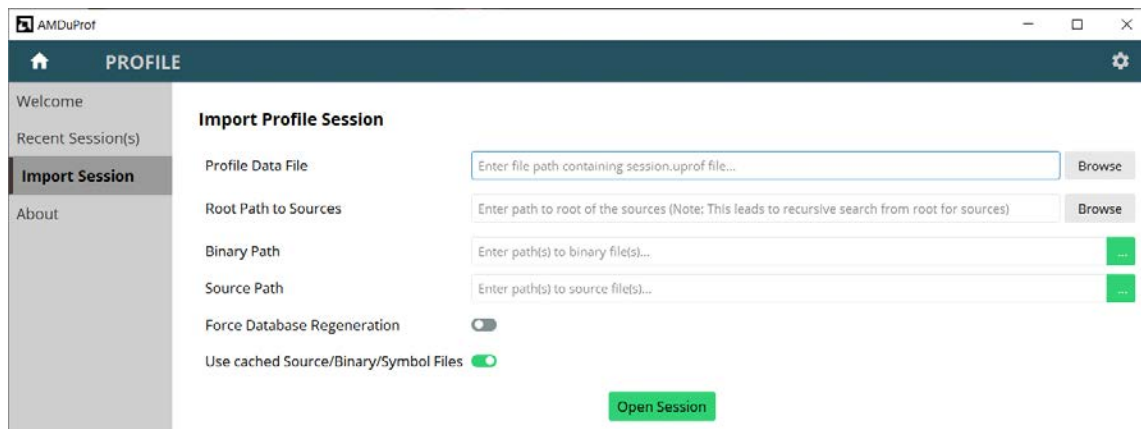


図 22. [Import Session] – プロファイル データベースのインポート

ここでインポートできるのは、CLI を使用して収集された処理済みプロファイル データか、GUI のプロファイル セッション保存パスに保存された処理済みプロファイル データです。インポート手順は次のとおりです。

- **[Profile Data File]** ボックスで、*session.uprof* ファイルを含むパスを指定します。
- **[Binary Path]**: プロファイリング実行がシステムで実行され、対応する生プロファイル データが別のシステムにインポートされた場合、バイナリ ファイルを含むパスを指定する必要があります。
- **[Source Path]**: ソース ファイルを含むソース パスを指定します。ソース ファイルを見つけるために、パス内のサブディレクトリが検索されることはありません。
- **[Root Path to Sources]**: 複数のソース ディレクトリのルートとなるパスを「指定します。そのパスに含まれるディレクトリおよびサブディレクトリ全体から、ソース ファイルが検索されます。

注記: すべてのサブディレクトリを再帰的に検索するため、検索に時間がかかる場合があります。

- **[Force Database Regeneration]**: インポート中、強制的にデータベース ファイルを再生成します。
- **[Use Cached Source/Binary/Symbol Files]**: キャッシュされたソース、バイナリ、シンボル ファイルを再利用するには、このオプションを有効にします。

5.7 保存済みのプロファイルセッションの解析

新しいプロファイルセッションを作成するか、プロファイルデータベースを開く (インポートする) と、履歴が更新されて、最後に開いたプロファイル データベース レコードが 50 件保存されます (履歴レコードはここに保存されています)。次に示すように、履歴リストは、[HOME] → [Recent Session(s)] から表示できます。

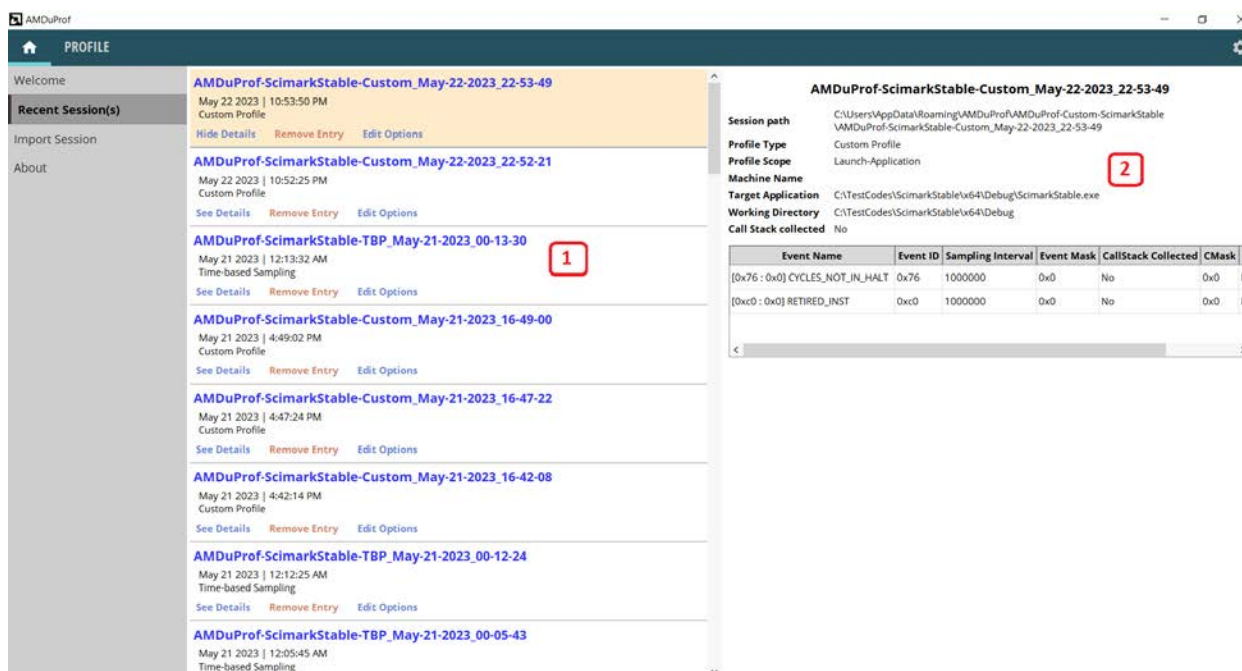


図 23. [PROFILE] - [Recent Session(s)]

上の画面の構成は次のとおりです。

- GUI で解析のために開いたプロファイルセッションの履歴。設定可能なオプションは次のとおりです。
 - いずれかのエントリをクリックすると、対応するプロファイルデータベースが解析用に読み込まれます。
 - [See Details] ボタンをクリックすると、プロファイル対象アプリケーション、監視対象イベント リストなど、このプロファイルセッションに関する詳細が表示されます。
 - [Edit Options] をクリックすると、セッションを開く前に、そのデータベースの [Import Profile Session] が自動的に入力され、必要な行編集ボックスが更新されます。
 - [Remove Entry] ボタンをクリックすると、現在のプロファイルセッションが履歴から削除されます。
- 選択されたプロファイルセッションの詳細を表示します。

5.8 保存されたプロファイル設定の使用

オプションを設定してプロファイリングを開始すると、プロファイル設定が作成されます。その後で、1つ以上の有効なプロファイル セッションが生成された場合、そのプロファイル設定の詳細が指定されたオプションと共に保存され、後から読み込むことができます。保存された設定のリストは、次に示すように、**[PROFILE]** → **[Saved Configurations]** から表示できます。

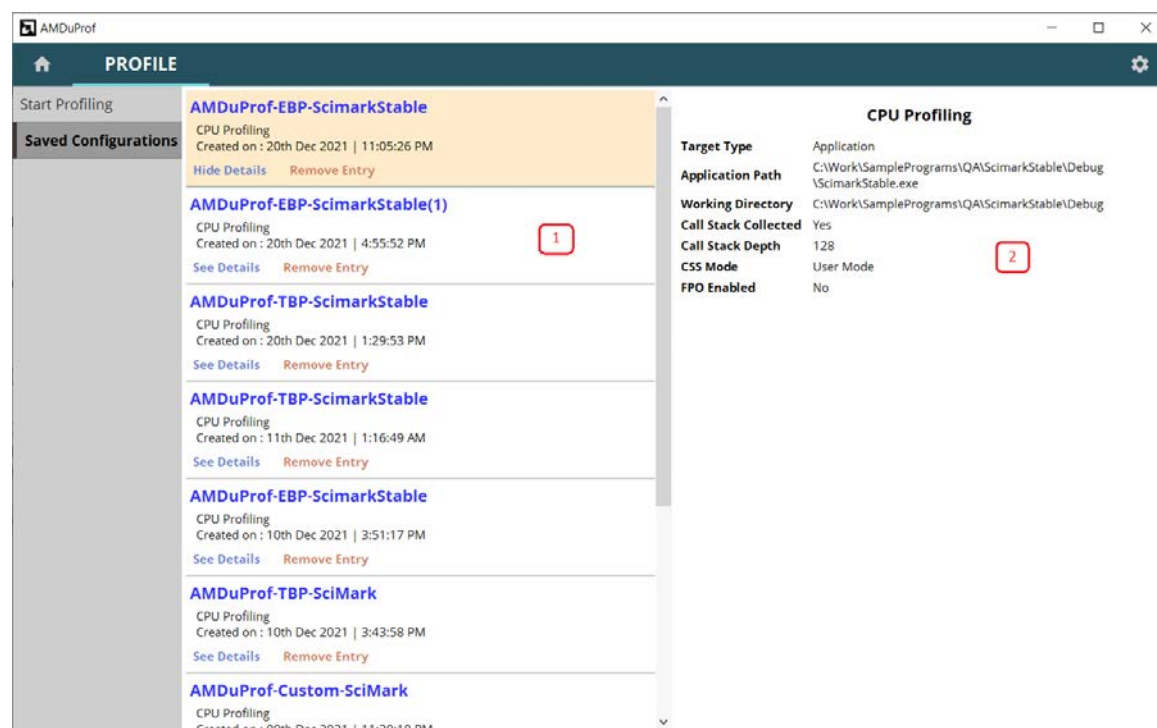


図 24. **[PROFILE]** - **[Saved Configurations]**

上の画面の構成は次のとおりです。

- GUI でのプロファイル データの収集に使用したプロファイル設定の履歴。設定可能なオプションは次のとおりです。
 - いずれかのエントリをクリックすると、対応するデータ収集用のプロファイル設定が表示されます。
 - [See Details]** ボタンをクリックすると、プロファイル対象アプリケーション、監視対象イベント リストなど、現在のプロファイル セッションに関する詳細が表示されます。
 - [Remove Entry]** ボタンをクリックすると、現在のプロファイル セッションが履歴から削除されます。
- 選択されたプロファイル セッションの詳細を表示します。

注記: デフォルトでは、プロファイル設定の名前はアプリケーションによって生成されます。再利用する場合は、簡単に見つけられるように適切な名前を付ける必要があります。名前を指定するには、**[PROFILE]** → **[Start Profiling]** の左下にある **[Config Name]** 行編集ボックスに設定名を入力します。

5.9 設定

AMD uProf のエクスペリエンスをカスタマイズするためのアプリケーション全体の設定があります。
[SETTINGS] ページは右上にあり、次に示す 3 つのセクションに分かれています。

- **[Preferences]:** このセクションを使用して、グローバル パスとデータ レポートのプリファレンスを設定します。

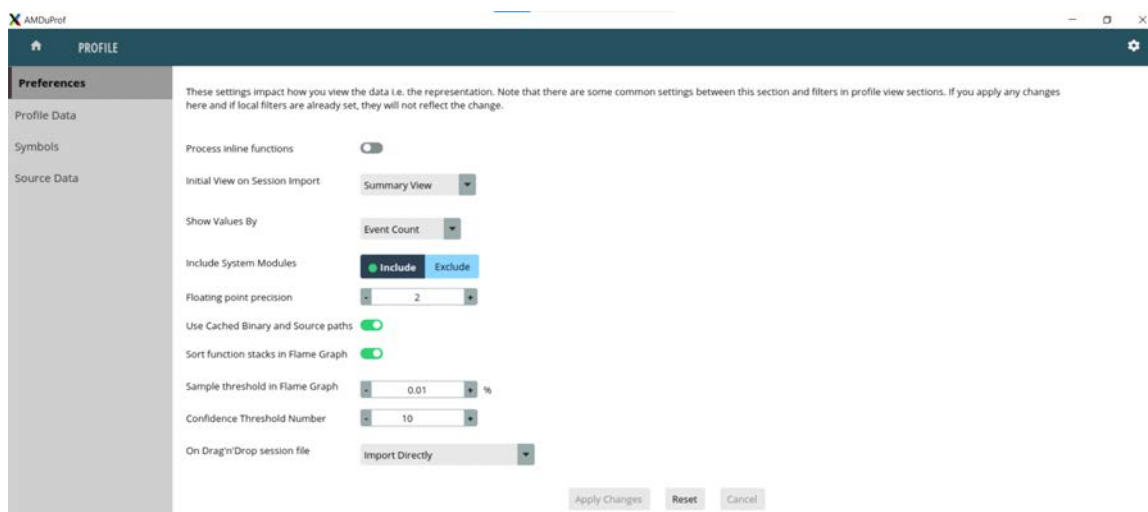


図 25. [SETTINGS] - [Preferences]

- **[Apply Changes]** ボタンをクリックすると、更新/変更した設定が適用されます。プロファイル データ フィルターには共通の設定があるため、**[Apply Changes]** ボタンで設定の変更が適用されるのは、ローカル フィルターが設定されていないビューのみです。
- **[Reset]** ボタンをクリックすると設定がリセットされ、**[Cancel]** ボタンをクリックすると、変更を適用せずに破棄できます。
- **[Symbols]:** このセクションを使用して、シンボル パスとシンボル サーバーのロケーションを設定します。シンボル サーバーは、Windows のみのオプションです。次の図に、**[Symbols]** セクションを示します。

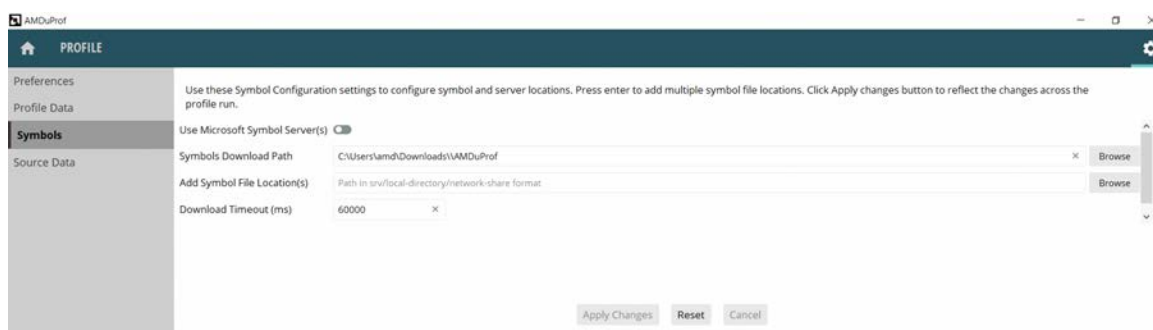


図 26. [SETTINGS] - [Symbols]

- **[Source Data]:** このセクションを使用して、ソース ビューのプリファレンスを設定します。次の図に、**[Source Data]** セクションを示します。

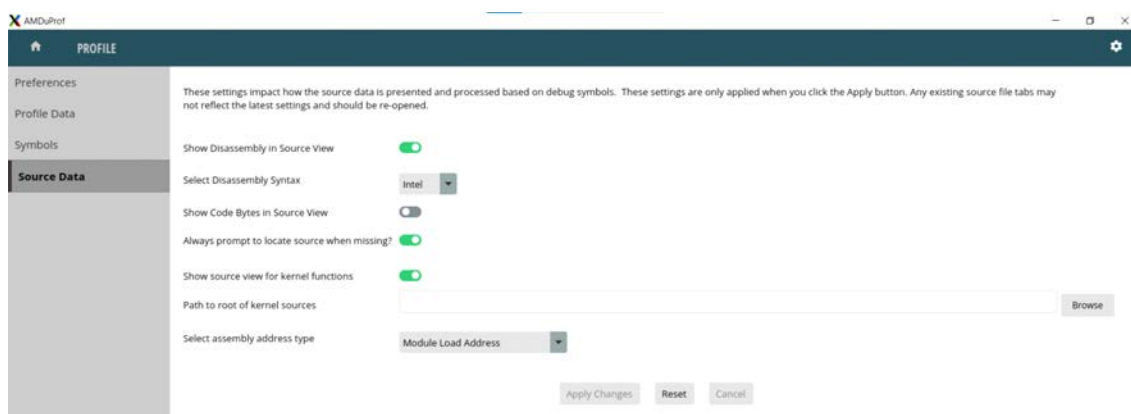


図 27. [SETTINGS] - [Source Data]

[Select Disassembly Syntax] を使用すると、ディスアセンブルの表示に使用する構文を選択できます。デフォルト設定で、Windows では Intel になっており、Linux では AT&T になっています。

- **[Profile Data]:** このセクションを使用して、プロファイリング中のデータ生成場所を指定します。次の図に、**[Profile Data]** セクションを示します。

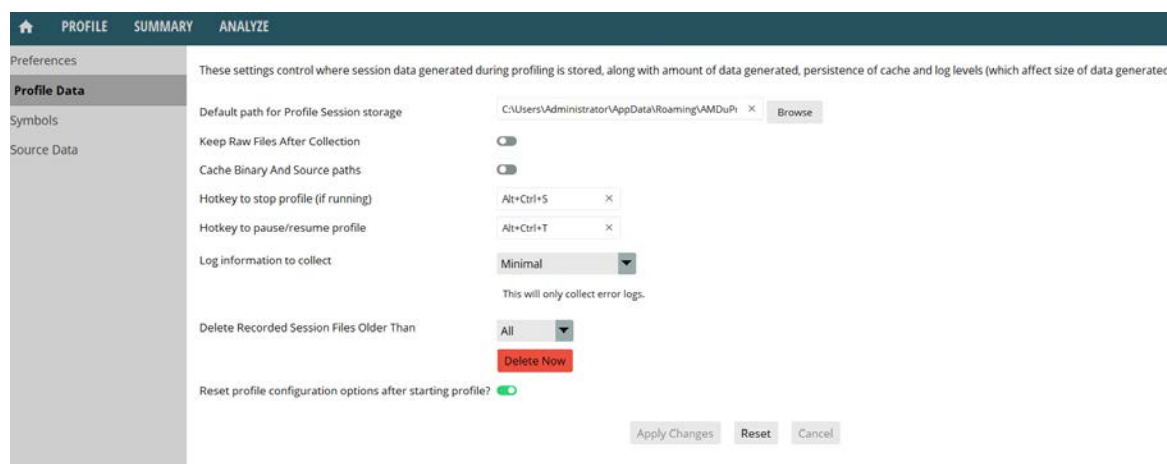


図 28. [Profile Data]

- **[Keep Raw Files After Collection]** をオンにすると、変換後に生ファイルが保存されます。デフォルトではオフになっています。
- **[Delete Record Session Files]** オプションを使用すると、指定した期間よりも古いセッション ファイルを削除できます。期間はデフォルトでは **[None]** に設定されています。
- **[Reset Profile Configuration]** を使用すると、各プロファイリング後にプリファレンスを追加するか、プロファイル設定をクリアするかを指定できます。デフォルトでは、**オン** (プロファイリング後にクリア) に設定されています。

- **[Hotkey to stop profile (if running)]** は、CPU および消費電力プロファイリングを停止するためのホットキーです。
- **[Hotkey to pause/resume profile]** は、CPU および消費電力プロファイリングを一時停止/再開するためのホットキーです。

注記: ホットキーがサポートされているのは Windows のみです。

5.10 ショートカット キー

次のテーブルに、AMD uProf のショートカット キーの一覧を示します。

表 23. ショートカット キー

ショートカット キー	説明
Ctrl + O	セッションをインポートします。
Ctrl + P	プロファイルの設定を開始し、プロファイルへのアプリケーション パスを指定します。
Ctrl + S	設定ページの最初のセクションにジャンプします。
Ctrl + F	[Function Hotspots] 画面で、検索バーにカーソルを移動します。これは、[Function Hotspots]、[Grouped Metrics]、[Flame Graph]、[Call Graph]、[Top-down]、[Callstack] で使用できます。
Ctrl + K	ソース ビューで、サンプル数が最大のソース行をハイライトします。
Ctrl +/-	タイムライン ビューをズーム イン/アウトします。
Ctrl + Z	タイムライン ビューの特定の領域にズームインします。

第 6 章 AMD uProf CLI の使用開始

6.1 概要

AMD uProf のコマンド ライン インターフェイスである AMDuProfCLI は、プロファイル データを解析するための収集およびレポート生成オプションを提供しています。

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

サポートされるコマンドは次のとおりです。

表 24. サポートされるコマンド

コマンド	説明
collect	指定されたプログラムを実行し、プロファイル サンプルを収集します。
report	生のプロファイル データファイル进行处理し、プロファイル レポートを生成します。
timechart	消費電力プロファイリング - 消費電力、熱、周波数メトリクスなどのシステム特性を収集してレポートします。
info	システムとトポロジに関する一般情報を表示します。
translate	生のプロファイル データファイル进行处理し、プロファイル データベースを生成します。
profile	パフォーマンス プロファイル データを収集して解析し、プロファイル レポートを生成します。
compare,diff	複数のプロファイル データ进行处理し、比較レポートを生成します。

ワークフローの詳細は、「[ワークフローと主な概念](#)」を参照してください。コマンド ライン インターフェイス AMDuProfCLI を実行するには、各 OS に応じて次のバイナリを実行します。

- Windows

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

- Linux

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

tar ファイルを使用してインストールしている場合

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

- FreeBSD

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

6.2 CPU プロファイルの開始

ネイティブ (C、C++、Fortran) アプリケーションのパフォーマンスをプロファイリングして解析するには、次の手順に従います。

1. アプリケーションを準備します。プロファイリングのためのアプリケーション準備の詳細は、「[参考資料](#)」を参照してください。
2. AMDuProfCLI の collect コマンドを使用して、アプリケーションのサンプルを収集します。

注記: FreeBSD で AMD uProf を実行する場合は、sudo コマンドからルート権限を使用します。

3. AMDuProfCLI の report コマンドを使用して、解析のために読みやすい形式でレポートを生成します。

アプリケーションの準備とは、起動アプリケーションをデバッグ情報付きで構築することです。デバッグ情報は、関数およびソース行にサンプルを関連付けるために必要になります。

collect コマンドを実行すると、アプリケーションが起動し (指定された場合)、指定されたプロファイル タイプおよびサンプリング設定に該当するプロファイル データが収集されます。生のデータ ファイル (Windows では .prd、FreeBSD では .pdata、Linux では .caperf) とその他の各種ファイルが生成されます。

report コマンドを実行すると、収集された生のプロファイル データが変換されて集計され、原因となるそれぞれのプロセス、スレッド、ロード モジュール、関数、命令が特定されます。また、データベースに情報が書き込まれ、CSV ファイル形式のレポートが生成されます。

次の図に、アプリケーション `AMDTClassicMatMul.exe` に対して時間ベースのプロファイリングを実行し、レポートを生成する方法を示します。

```
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>
```

図 29. Collect コマンドと Report コマンド

6.2.1 事前定義サンプル設定のリスト

collect コマンドの **--config** オプションでの使用がサポートされる事前定義サンプリング設定の一覧を表示するには、次のコマンドを実行します。

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

次に出力サンプルを示します。

```
[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.
```

図 30. Linux でサポートされる事前定義設定

```
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
```

図 31. Windows でサポートされる事前定義設定

6.2.2 プロファイル レポート

CSV 形式のプロファイル レポートには、次のセクションが含まれます。

- EXECUTION — ターゲットの起動アプリケーションに関する情報。
- PROFILE DETAILS — プロファイル タイプ、スコープ、サンプリング イベントなど、現在のセッションに関する情報。
- MONITORED EVENTS — プロファイル対象イベントと対応するサンプリング間隔のリスト。
- 10 HOTTEST FUNCTIONS — 上位 10 のホットな関数と、これらの関数に起因するメトリクス of the リスト。
- TAKEN BRANCH ANALYSIS SUMMARY — 上位 10 のホットな分岐のリスト。

- 10 HOTTEST PROCESSES — 上位 10 のホットなプロセスと、これらのプロセスに起因するメトリクスのリスト。
- 10 HOTTEST MODULES — 上位 10 のホットなモジュールと、これらのモジュールに起因するメトリクスのリスト。
- 10 HOTTEST THREADS — 上位 10 のホットなスレッドと、これらのスレッドに起因するメトリクスのリスト。
- PROFILE REPORT FOR PROCESS — プロファイル対象プロセスに起因するメトリクス。このセクションは、レポート生成で `--detail option` が使用された場合に表示されます。次のようなサブセクションが含まれます。
 - THREAD SUMMARY — スレッドと、スレッドに起因するメトリクスのリスト。
 - MODULE SUMMARY — プロセスに属するロード モジュールと、これらのモジュールに起因するメトリクスのリスト。
 - FUNCTION SUMMARY — サンプル収集の対象となったプロセスに属する関数と、これらの関数に起因するメトリクスのリスト。
 - LAST BRANCH RECORD FOR PROCESS — このプロセスに対して収集された分岐のリスト。
 - Function Detail Data — サンプル収集の対象となった上位関数のソース レベルでの特定。
 - CALLGRAPH — コールグラフ (コールスタック サンプルが収集された場合)。

6.3 消費電力プロファイルの開始

6.3.1 システム全体の消費電力プロファイル (ライブ)

消費電力プロファイルのカウンター値を収集するには、次の手順に従います。

1. AMDuProfCLI の `timechart` コマンドに `--list` オプションを指定して実行すると、サポートされるカウンター カテゴリの一覧が表示されます。
2. AMDuProfCLI の `timechart` コマンドを使用し、`--event` オプションで目的のカウンターを指定して、必要なカウンターを収集し、レポートします。

次のように、`timechart` を実行して、サポートされるカウンター カテゴリのリストを表示します。

```
[amd@win-f7f2kcsgh7k 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]

図 32. timechart --list コマンドの出力

次のように、timechart を実行してプロファイル サンプルを収集し、ファイルに書き込みます。

```
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>

図 33. timechart の実行

上記に示した実行により、対象カウンターがサポートされるすべてのデバイスで、消費電力と周波数のカウンターが収集され、**-o** オプションで指定された出力ファイルに書き込まれます。プロファイリングの開始前に、指定されたアプリケーションが起動して、アプリケーションが終了するまでの間、データが収集されます。

6.4 収集コマンド

collect コマンドを実行すると、パフォーマンス プロファイル データが収集され、指定された出力ディレクトリ内で生データ ファイルに書き込まれます。これらのファイルは、後から、AMDuProfCLI の report コマンドまたは AMDuProf GUI を使用して解析できます。

構造:

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

<PROGRAM> — プロファイリングの対象となる起動アプリケーションを指定します。

<ARGS> — 起動アプリケーションに渡す引数のリストを指定します。

一般的な使用法:

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

6.4.1 オプション

次の表に、収集コマンドのオプションの一覧を示します。

表 25. AMDuProfCLI の Collect コマンドのオプション

オプション	説明
-h --help	ヘルプ情報をコンソール/ターミナルに表示します。
-o --output-dir <directory-path>	収集されたデータ ファイルが保存されるベースディレクトリのパス。このディレクトリ内に、新しいサブディレクトリが作成されます。
--config <config>	サンプル収集で使用する事前定義サンプリング設定。 サポートされる設定のリストを表示するには、 <code>info --list collect-configs</code> コマンドを使用します。--config は複数指定できます。

表 25. AMDuProfCLI の Collect コマンドのオプション (続き)

オプション	説明
-e --event or <predefined-event>	-e、--event では、引数が事前に定義された事前定義イベントを直接使用できます。 または、より詳細なパラメーターを指定する方法として、タイマー、PMU、IBS イベント、または事前定義イベントを、キー=値ペアのカンマ区切り形式で引数に指定できます。サポートされているキーは次のとおりです。 - event=<timer ibs-fetch ibs-op> または <PMU-event> または <predefined-event> - umask=<unit-mask> - user=<0 1> - os=<0 1> - cmask=<count-mask> (値の範囲は 0x0 から 0x7f) - inv=<0 1> - interval=<sampling-interval> - frequency=<frequency (n)> (コア PMC イベントのみでサポート。周波数は Hz で指定) - ibsop-count-control=<0 1> (ibs-op イベントの場合) - loadstore (ibs-op イベントの場合。Windows プラットフォームのみ) - ibsop-l3miss (IBS オペレーション イベントの場合。AMD “Zen4” プロセッサのみでサポート) - ibsfetch-l3miss (IBS フェッチ イベントの場合。AMD “Zen4” プロセッサのみでサポート) - call-graph 注記: - 事前定義イベントの場合、umask の指定は必須ではありません。 - コールスタック サンプルの収集に関連する引数を指定するには、専用オプション --call-graph を使用します。 引数の詳細は、次のとおりです。 - user – ユーザー スペースのサンプル収集を有効 (1) または無効 (0) にします。 - os – カーネル スペースのサンプル収集を有効 (1) または無効 (0) にします。 - interval – サンプル収集の間隔。タイマーでは、時間間隔をミリ秒で指定します。PMU と事前定義イベントの場合は、イベント発生回数です。IBS フェッチの場合はフェッチ回数です。IBS オペレーションの場合、サイクル数またはディスパッチ数になります。 - op-count-control – IBS オペレーションのサンプリング数を、サイクル数 (0) またはディスパッチ数 (1) から選択します。 - loadstore – IBS オペレーションのロード/ストア サンプル収集だけを有効にします。その他の IBS オペレーション サンプルは収集されません。 - ibsop-l3miss – l3 ミス発生時のみ、IBS オペレーションのサンプル収集を有効にします。 例: 「-e event=ibs-op,interval=100000,ibsop-l3miss」

表 25. AMDuProfCLI の Collect コマンドのオプション (続き)

オプション	説明
	• <code>ibsfetch-l3miss -l3</code> ミス発生時のみ、IBS フェッチのサンプル収集を有効にします。 例: 「<code>-e event=ibs-fetch,interval=100000,ibsfetch-l3miss</code>」 引数が指定されない場合のデフォルト値は次のとおりです。 • <code>umask=0</code> • <code>cmask=0x0</code> • <code>user=1</code> • <code>os=1</code> • <code>inv=0</code> • <code>ibsopt-count-control=0</code> (ibs-op イベントの場合) • <code>interval=1.0 ms</code> (timer イベントの場合) • <code>interval=250000</code> (ibs-fetch、ibs-op、pmu-event、predefined-event の場合) 必要に応じて次のコマンドを使用します。 • <code>info --list predefined-events</code>: サポートされる事前定義イベントのリスト • <code>info --list pmu-events</code>: サポートされる PMU イベントのリスト <code>--event (-e)</code> は複数指定できます。
<code>-p --pid <PID...></code>	特定の実行中プロセスに接続することで、既存プロセスをプロファイリングします。プロセス ID の区切りにはカンマを使用します。 注記: 一度に最大で 512 のプロセスに接続できます。
<code>-a --system-wide</code>	システム全体のプロファイル (SWP) このフラグが設定されていない場合、コマンドライン ツールでプロファイリングされるのは、起動したアプリケーションか、<code>-p</code> オプションで接続したプロセス ID のみになります。
<code>-c --cpu <core...></code>	プロファイリングする CPU のカンマ区切りリスト。「-」を使用すると、CPU の範囲を指定できます。例: <code>0-3</code>。 注記: Windows では、選択されたコアが 1 つのプロセッサグループのみに属している必要があります。たとえば、<code>0-63</code>、<code>64-127</code> などです。
<code>-d --duration <n></code>	指定された時間 (秒) だけプロファイリングを実行します。
<code>--interval <num></code>	PMC イベントのサンプリング間隔。 注記: この間隔によって、個別イベントに指定されたサンプリング間隔がオーバーライドされます。
<code>--affinity <core...></code>	プロファイル対象となる起動アプリケーションのコアのアフィニティを設定します。コア ID のカンマ区切りリスト。コア ID の範囲を指定する必要があります。例: <code>0-3</code>。デフォルト アフィニティは、使用可能なすべてのコアです。
<code>--no-inherit</code>	起動アプリケーションの子 (プロファイル対象アプリケーションによって起動されるプロセス) はプロファイリングしません。
<code>-b --terminate</code>	プロファイル データの収集が終了した後で、起動アプリケーションを終了します。中止するのは起動アプリケーションのプロセスのみです。子プロセスがある場合、その実行は継続される可能性があります。
<code>--start-delay <n></code>	<code>n</code> 秒の開始遅延。指定された時間の経過後にプロファイリングを開始します。<code>n</code> が 0 の場合は何も影響はありません。

表 25. AMDuProfCLI の Collect コマンドのオプション (続き)

オプション	説明
--start-paused	プロファイリングが無期限に一時停止します。ターゲット アプリケーションが、プロファイル制御 API を使用してプロファイリングを再開します。このオプションは、再開/一時停止 API (「 AMDProfileControl API 」参照) を使用したプロファイル データ収集の制御が、起動アプリケーションにインストルメント化されている場合のみ使用してください。
-w --working-dir <path>	作業ディレクトリを指定します。デフォルトは、現在の作業ディレクトリです。
--log-path <path-to-log-dir>	ログ ファイルが作成される場所へのパスを指定します。このオプションを指定しない場合、デフォルトでログ ファイルが作成される場所は、AMDUPROF_LOGDIR 環境変数で設定されたパス、\$TEMP パス (Linux、FreeBSD)、または %TEMP% パス (Windows) になります。 ログ ファイル名の形式は、\$USER-AMDuProfCLI.log (Linux、FreeBSD) または %USERNAME%-AMDuProfCLI.log (Windows) となります。
--enable-log	ログ ファイルへの追加ロギングを有効にします。
--enable-logts	ログ レコードのタイムスタンプを取り込みます。--Enable-log オプションと一緒に使用します。
--limit-size <n>	収集データのファイル サイズ (MB) が指定された上限を超えた場合、プロファイリングを停止します。 注記: このオプションは、今後のリリースで廃止される可能性があります。
--frequency <n> --freq <n> -F <n>	コア PMC イベントに対して、指定された周波数「n」(Hz) でのデータ収集を有効にします。 注記: この周波数によって、個別イベントに指定されたサンプリング周波数がオーバーライドされます。

6.4.2 Windows 専用オプション

次の表に Linux 専用の収集コマンド一覧を示します。

表 26. AMDuProfCLI の Collect コマンド – Windows 専用オプション

オプション	説明
--call-graph <I:D:S:F>	コールスタック サンプリングを有効にします。アンワインド間隔 (I) をミリ秒で指定し、アンワインドの深さ (D) の値を指定します。スコープ (S) を指定するには、次のいずれかを選択します。 - user: ユーザー スペースのコードのみを収集対象とします。 - kernel: カーネル スペースのコードのみを収集対象とします。 - all: ユーザー空間とカーネル空間で実行されるコードを収集対象とします。 コンパイラによるフレーム ポインターの省略 (F) が原因で見つからないフレームの収集方法を指定します。 - fpo: フレーム ポインターが使用できない場合、アンワインド情報を使用してコールスタック情報を収集します。 - fp: フレーム ポインターを使用して、コールスタック情報を収集します。
-g	次を指定するのと同じです。--call-graph 1:128:user:fp.
--thread <thread=concurrency>	ランタイム スレッドの詳細を収集します。

表 26. AMDuProfCLI の Collect コマンド – Windows 専用オプション (続き)

オプション	説明
<code>-m --data-buffer-count <size></code>	ドライバによるデータ収集に使用されるバッファのサイズ (コアあたりのページ数) を指定します。デフォルト サイズは、コアあたり 512 ページです。
<code>--trace os</code>	ターゲットのドメイン OS をトレースします。サポートされるのは、「スケジュール イベント」のみです。OS トレース イベントの一覧を表示するには、' <code>info --list trace-events</code> ' コマンドを使用します。
<code>--limit-data <n></code>	収集データのファイル サイズ (MB) が指定された上限を超えた場合、プロファイリングを停止します。 <code>--overwrite</code> オプションと一緒に使用した場合、上限は収集が停止される前になります。サイズは、メガバイト (M/m)、ギガバイト (G/g)、または秒 (secs) を末尾に付けて指定できます。
<code>--overwrite</code>	プロファイル データ収集モードをリング バッファとして設定します。収集の上限は、オプション <code>--limit-data</code> を使用して設定できます。デフォルトの <code>--limit-data</code> では、生のデータ ファイル サイズがコアあたり 512 ページに制限されます。

6.4.3 Linux 専用オプション

次の表に Linux 専用の収集コマンド一覧を示します。

表 27. AMDuProfCLI の Collect コマンド – Linux 専用オプション

オプション	説明
<code>--call-graph <F:N></code>	コールスタック サンプリングを有効にします。コンパイラによるフレーム ポインターの省略が原因で見つからないフレームを収集するか、無視するかを指定 (F) します。 <code>fpo</code> <code>dwarf</code>: サンプル収集中にプロセス コールスタックを収集し、DWARF 情報を使用してコールスタックを再構築します。 <code>fp</code>: フレーム ポインターを使用して、コールスタック情報を収集します。 F = <code>fpo</code> の場合、「N」によって、1 回のサンプル収集で収集する最大スタック サイズがバイトで指定されます。スタック サイズの有効な範囲は、16 ~ 32768 です。「N」が 8 の倍数でない場合、最も近い 8 の倍数まで引き下げられます。デフォルト値は 1024 バイトです。 注記: 大きい N 値を指定すると、生成される生データ ファイルが非常に大きくなります。 F = <code>fp</code> の場合、「N」の値は無視されるため、指定する必要はありません。
<code>-g</code>	<code>--call-graph fp</code> を指定するのと同じです。
<code>--tid <TID,...></code>	特定の実行中スレッドに接続することで、既存スレッドをプロファイリングします。スレッド ID の区切りにはカンマを使用します。

表 27. AMDuProfCLI の Collect コマンド – Linux 専用オプション (続き)

オプション	説明
<code>--trace <TARGET></code>	ターゲット ドメインをトレースします。TARGET には、次に示すオプションから 1 つ以上を指定します。 <code>mpi[=<openmpi mpich>,<lwt full>]</code> 次のとおりに MPI インプリメンテーション タイプを指定します。 OpenMPI ライブラリをトレースする場合、「openmpi」 Intel MPI など、MPICH とその派生ライブラリをトレースする場合、「mpich」 次のとおりにトレースの範囲を指定します。 軽量トレースの場合、「lwt」 完全なトレースの場合、「full」 「--trace mpi」は、デフォルトで「--trace mpi=mpich,full」となります。 <code>openmp</code> — OpenMP アプリケーションをトレースする場合。これは、<code>--omp</code> オプションと同じです。 <code>os[=<event1,event2,...>]</code> — イベント名とオプションのしきい値をカンマ区切りリストで指定します。syscall イベントと memtrace イベントは、オプションのしきい値を <code><event:threshold></code> として受け取ります。OS トレース イベントの一覧を表示するには、<code>info --list trace-events</code> コマンドを使用します。 <code>user[=<event1,event2,...>]</code> — イベント名としきい値をカンマ区切りリストで指定します。これらのイベントはユーザー モードで収集されます。ユーザー モードでサポートされるトレース イベントの一覧を表示するには、<code>info --list trace-events</code> コマンドを使用します。 <code>gpu[=<hip,hsa>]</code> — GPU トレースのドメインを指定します。デフォルトでは、ドメインは「hip,hsa」に設定されています。
<code>--buffer-size <size></code>	OS トレース バッファに割り当てるページ数。デフォルト値は、コアあたり 256 ページです。トレース データの損失を少なくするには、ページ数を増やします。このオプションを使用できるのは、OS トレース (<code>--trace os</code>) の場合のみです。
<code>--max-threads <thread-count></code>	OS トレースの対象とする最大スレッド数。起動アプリケーションの場合、デフォルト値は 1024 です。システム全体のトレース (<code>-a</code> オプション) の場合は 32768 です。アプリケーションのスレッド数がデフォルト制限を超えた場合は、この制限値を増やします。そうしない場合、動作が不定になります。 - 起動アプリケーション - 有効な範囲: 1 ~ 4096 - システム全体 - 有効な範囲: 1 ~ 4194304
<code>--func <module:function-pattern></code>	トレースする関数を、ライブラリ、実行ファイル、またはカーネルから指定します。 <code>function-pattern</code> には、関数名か、「*」で終わる部分的な名前を指定できます。「*」のみを指定すると、モジュールに含まれるすべての関数がトレースされます。 <code>module</code> にはライブラリまたは実行ファイルを指定します。カーネル関数をトレースする場合は、<code>module</code> を「kernel」で置換します。 注記: モジュールの絶対パス/フルパスを指定することを推奨します。それ以外の場合は、現在の作業ディレクトリではなく、デフォルト ライブラリ パスに対して検索が実行されます。

表 27. AMDuProfCLI の Collect コマンド – Linux 専用オプション (続き)

オプション	説明
--exclude-func <module:function-pattern>	除外する関数を、ライブラリ、実行ファイル、またはカーネルから指定します。 - function-pattern には、関数名か、「*」で終わる部分的な名前を指定できます。「*」のみを指定すると、モジュールに含まれるすべての関数がトレースされます。 - module にはライブラリまたは実行ファイルを指定します。カーネル関数をトレースする場合は、module を「kernel」で置換します。 注記: モジュールの絶対パスを指定することを推奨します。それ以外の場合は、現在の作業ディレクトリではなく、デフォルトライブラリパスに対して検索が実行されます。
-m --mmap-pages <size>	カーネル メモリ マップド データ バッファのサイズを設定します。サイズは、ページ数で指定するか、バイト (B/b)、キロバイト (K/k)、メガバイト (M/m)、ギガバイト (G/g) を末尾に付けて指定できます。
--mpi	MPI アプリケーションの CPU プロファイリング データを収集する際、このオプションを指定します。MPI トレースの場合、--trace オプションを参照してください。
--kvm-guest <pid>	guest サイドのパフォーマンス プロファイルを収集するために、プロファイリングする qemu-kvm プロセスの PID を指定します。
--guest-kallsyms <path>	ローカル ホストにコピーされた guest/proc/kallsyms のパスを指定します。AMD uProf はこれを読み取って、ゲスト カーネルシンボルを取得します。
--guest-modules <path>	ローカル ホストにコピーされた guest/proc/modules のパスを指定します。AMD uProf はこれを読み取って、ゲスト カーネルのモジュール情報を取得します。
--guest-search-path <path>	ローカル ホストにコピーされたゲスト vmlinux とカーネル ソースのパスを指定します。AMD uProf はこれを読み取って、ゲスト カーネルのモジュール情報を解決します。
--branch-filter	LBR データを取り込みます。 分岐フィルター タイプも指定できます。 - u: ユーザー分岐 - k: カーネル分岐 - any: 任意の分岐タイプ - any_call: 任意の呼び出し分岐 - any_ret: 任意のリターン分岐 - ind_call: 間接呼び出し - ind_jump: 間接ジャンプ - cond: 条件付き分岐 - call: 直接呼び出し 注記: - 上記のフィルターが設定されていない場合、デフォルトのフィルター タイプは「any」になります。 - このオプションは、PMC イベントのみで正しく動作します。 - これはプロセスごとに適用され、プロセス プロファイリングに関連付けられます。ただし、Java アプリケーションのプロファイリングには適用できません。

6.4.4 例

Windows

- アプリケーション *AMDTClassicMatMul.exe* を起動し、CYCLES_NOT_IN_HALT および RETIRED_INST イベントのサンプルを収集します。

```
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
```

- AMDTClassicMatMul.exe* アプリケーションを起動し、時間ベース プロファイル (TBP) のサンプルを収集します。

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

- AMDTClassicMatMul.exe* を起動し、Assess Performance のプロファイリングを 10 秒間実行します。

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

- AMDTClassicMatMul.exe* を起動し、SWP モードで IBS サンプルを収集します。

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

- SWP モードで 10 秒間にわたって TBP サンプルを収集します。

```
C:\> AMDuProfCLI.exe collect -a -o c:\Temp\cpuprof-tbp-swp -d 10
```

- AMDTClassicMatMul.exe* を起動し、コールスタック サンプリング付きで TBP を収集します。

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

- AMDTClassicMatMul.exe* を起動し、コールスタック サンプリング付きで TBP を収集します (アンワインド FPO 最適化スタック)。

```
C:\> AMDuProfCLI.exe collect --config tbp --call-graph 1:64:user:fpo -o c:\Temp\cpuprof-tbp
AMDTClassicMatMul.exe
```

- AMDTClassicMatMul.exe* を起動し、PMCx076 および PMCx0C0 のサンプルを収集します。

```
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
```

- AMDTClassicMatMul.exe* を起動し、50000 の間隔で IBS オペレーションのサンプルを収集します。

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

- AMDTClassicMatMul.exe* を起動し、スレッドの同時並行性および名前に関する TBP サンプルプロファイリングを実行します。

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

- AMDTClassicMatMul.exe* を起動し、SWP モードで消費電力サンプルを収集します。

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

- PMCx076 および PMCx0C0 のサンプルを収集しますが、コールグラフ情報は PMCx0C0 のみに対して収集します。

```
C:\> AMDuProfCLI.exe collect -e event=pmcx76,interval=250000 -e
event=pmcxc0,interval=250000,call-graph -o c:\Temp\cpuprof-pmc AMDTClassicMatMul-bin
```

- AMDTClassicMatMul.exe を起動し、事前定義イベント RETIRED_INST と L1_DC_REFILLS.ALL のサンプルを収集します。

```
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
```

- AMDTClassicMatMul.exe を起動し、TBP および Assess Performance のサンプルを収集します。

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

- Mutithread_Threadname.exe を起動し、スケジュール イベントを収集します。

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

- AMDTClassicMatMul.exe を起動し、カウント マスクを有効にして PMCx076 および PMCx0C0 イベントのサンプルを収集します。

```
C:\> AMDuProfCLI.exe collect -e event=pmcx076,cmask=0x0, -e
event=pmcxc0,cmask=0x7f,interval=250000 -o c:\Temp\cpuprof-pmc AMDTClassicMatMul-bin
```

Linux

- アプリケーション AMDTClassicMatMul.bin を起動し、CYCLES_NOT_IN_HALT および RETIRED_INST イベントのサンプルを収集します。

```
$ ./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
```

- アプリケーション AMDTClassicMatMul-bin を起動し、TBP サンプルを収集します。

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

- AMDTClassicMatMul-bin を起動し、Assess Performance のプロファイリングを 10 秒間実行します。

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

- AMDTClassicMatMul-bin を起動し、SWP モードで IBS サンプルを収集します。

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

- SWP モードで 10 秒間にわたって TBP サンプルを収集します。

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

- AMDTClassicMatMul-bin を起動し、コールスタック サンプリング付きで TBP を収集します。

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

- *AMDTClassicMatMul-bin* を起動し、コールスタック サンプリング付きで TBP を収集します (アンワインド FPO 最適化スタック)。

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

- *AMDTClassicMatMul-bin* を起動し、PMCx076 および PMCx0C0 のサンプルを収集します。

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

- *AMDTClassicMatMul-bin* を起動し、50000 の間隔で IBS オペレーションのサンプルを収集します。

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

- スレッドに接続して、TBP サンプルを 10 秒間にわたって収集します。

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

- OpenMP アプリケーションの OpenMP トレース情報を収集し、--omp を渡します。

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

- *AMDTClassicMatMul-bin* を起動し、キャッシュの偽共有に関してメモリ アクセスを収集します。

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

- *AMDTClassicMatMul-bin* を起動し、スレッド間でホットスポット、スレッド ステートを、待機オブジェクトを解析するために、スレッド設定を収集します

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

- MPI プロファイリング情報を収集します。

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

- PMCx076 および PMCx0C0 のサンプルを収集しますが、コールグラフ情報は PMCx0C0 のみに対して収集します。

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

- *AMDTClassicMatMul-bin* を起動し、事前定義イベント RETIRED_INST と L1_DC_REFILLS.ALL のサンプルを収集します。

```
$ 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
```

- *AMDTClassicMatMul-bin* を起動し、すべての OS トレース イベントを収集します。

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

- *AMDTClassicMatMul-bin* を起動し、すべてのユーザー モードのトレース イベントを収集します。

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

- *AMDTClassicMatMul-bin* を起動し、1 μs 以上かかっている syscall を収集します。

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

- *AMDTClassicMatMul-bin* を起動し、hip ドメインの GPU トレースを収集します。

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

- *AMDTClassicMatMul-bin* を起動し、hip および hsa ドメインの GPU トレースを収集します。
\$ AMDuProfCLI collect --trace gpu -o /tmp/cpuprof-gpu AMDTClassicMatMul-bin
- *AMDTClassicMatMul-bin* を起動し、hip ドメインの TBP サンプルと GPU トレースを収集します。
\$ AMDuProfCLI collect --config tbp --trace gpu=hip -o /tmp/cpuprof-gpu AMDTClassicMatMul-bin
- *AMDTClassicMatMul-bin* を起動し、GPU サンプルを収集します。
\$ AMDuProfCLI collect --config gpu -o /tmp/cpuprof-gpu AMDTClassicMatMul-bin
- *AMDTClassicMatMul-bin* を起動し、GPU サンプルと OS トレースを収集します。
\$ AMDuProfCLI collect --config gpu --trace os -o /tmp/cpuprof-gpu-os AMDTClassicMatMul-bin
- *AMDTClassicMatMul-bin* を起動し、TBP および GPU サンプルを収集します。
\$ AMDuProfCLI collect --config gpu --config tbp -o /tmp/cpuprof-gpu-tbp AMDTClassicMatMul-bin
- *AMDTClassicMatMul-bin* を起動し、呼び出された malloc() の関数カウントを収集します。
\$ AMDuProfCLI collect --trace os=funccount --func c:malloc -o /tmp/cpuprof-os AMDTClassicMatMul-bin
- *AMDTClassicMatMul-bin* を起動し、コンテキスト スイッチ、syscall、pthread API トレース、呼び出された malloc() の関数カウントを収集します。
\$ AMDuProfCLI collect --trace os --func c:malloc -o /tmp/cpuprof-os AMDTClassicMatMul-bin
- malloc()、calloc()、カーネル関数のうち、パターン「vfs_read*」に一致する関数カウントをシステム全体で収集します。
\$ AMDuProfCLI collect --trace os --func c:malloc,calloc,kernel:vfs_read* -o /tmp/cpuprof-os -a -d 10
- *AMDTClassMatMul-bin* を起動し、デフォルトのフィルター タイプを使用して分岐解析を実行します。
\$ AMDuProfCLI collect --branch-filter -o /tmp/cpuprof-ebp-branch AMDTClassicMatMul-bin
- *AMDTClassMatMul-bin* を起動し、イベント PMCXC0 のサンプルを収集します。
\$ AMDuProfCLI collect -e event=pmcxc0,interval=250000 --branch-filter u,k,any -o /tmp/cpuprof-ebp-branch AMDTClassicMatMul-bin

6.5 レポート コマンド

report コマンドを使用すると、生のプロファイル データを処理するか、指定されたディレクトリにある (処理済みの) データベース ファイルを使用して、読みやすい形式でレポートを生成できます。

構造:

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

一般的な使用法:

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

6.5.1 オプション

表 28. AMDuProfCLI の report コマンドのオプション

オプション	説明
-h --help	このヘルプ情報をコンソール/ターミナルに表示します。
-i --input-dir <directory-path>	収集されたデータを含むディレクトリへのパス。
--detail	詳細なレポートを生成します。
--group-by <section>	生成されるレポートを指定します。サポートされるレポート オプションは次のとおりです。 • process: プロセスの詳細をレポート • module: モジュールの詳細をレポート • thread: スレッドの詳細をレポート このオプションが適用されるのは、--detail オプションと共に使用する場合があります。デフォルトは、 group-by process です。
-p --pid <PID,...>	指定された PID のレポートを生成します。プロセス ID の区切りにはカンマを使用します。 注記: 一度に最大で 512 のプロセスに接続できます。
-g	コールグラフを出力します。--detail オプションまたは --pid(-p) オプションと一緒に使用します。--pid オプションを使用する場合、指定された PID のコールスタック サンプルが収集されている場合のみ、コールグラフが生成されます。
--cutoff <n>	レポート対象となるプロセス、スレッド、モジュール、関数を制限するためのカットオフ。n は、各種レポート セクションに記載されるエントリの最小数です。デフォルト値は 10 です。
--view <view-config>	指定されたビュー ファイルに含まれるイベントのみをレポートします。サポートされるビュー設定の一覧を表示するには、info --list view-configs コマンドを使用します。
--inline	C、C++ 実行ファイルのインライン関数を表示します。 注記: 1. このオプションは Windows ではサポートされていません。 2. このオプションを使用すると、レポート生成にかかる時間が長くなります。
--show-sys-src	システム モジュール関数の詳細な関数レポートを、(デバッグ情報を使用できる場合は) ソース ステートメント付きで生成します。
--src-path <path1;...>	ソース ファイルのディレクトリ (複数のパスをセミコロンで区切ります)。 --src-path は複数使用できます。
--disasm	詳細な関数レポートをアセンブリ命令付きで生成します。
--disasm-style <att intel>	アセンブリ命令の構文を選択します。サポートされるオプションは、att と intel です。このオプションを使用しない場合は、次のとおりとなります。 • Windows では、デフォルトで intel が使用されます。 • Linux では、デフォルトで att が使用されます。
--disasm-only	アセンブリ命令だけを含む関数レポートを生成します。

表 28. AMDuProfCLI の report コマンドのオプション

オプション	説明
-s --sort-by <EVENT>	レポート対象プロファイル データを並べ替える基準として、タイマー、PMC、または IBS イベントを、キー=値ペアのカンマ区切り形式で引数として指定します。サポートされているキーは次のとおりです。 • event=<timer ibs-fetch ibs-op pmcxNNN>。ここで、NNN は 16 進数のコア PMC イベント ID です。 • umask=<unit-mask> • cmask=<count-mask> • inv=<0 1> • user=<0 1> • os=<0 1> サポートされる PMC イベントの一覧を表示するには、<code>info --list pmu-events</code> コマンドを使用します。 引数の詳細は次のとおりです。 • umask — 10 進数または 16 進数のユニット マスク。PMC イベントのみに適用されます。 • cmask — 10 進数または 16 進数のカウント マスク。PMC イベントのみに適用されます。 • user、os — ユーザー モードと OS モード。PMC イベントのみに適用されます。 • inv — 逆カウント マスク。PMC イベントのみに適用されます。 -sort-by (-s) を複数指定することはできません。
--agg-interval <low medium high INTERVAL>	このオプションを使用して、サンプル集計の間隔を設定します。これは、セッションを GUI にインポートするときに便利です。 low レベルの集計間隔を指定すると、GUI のタイムライン ビューの品質は高くなりますが、データベース サイズが大きくなります。 INTERVAL には集計間隔の数値をミリ秒で指定できます。
--time-filter <T1:T2>	レポート生成時間を T1 と T2 の間に限定します。ここで、T1 と T2 はプロファイル開始時間からの秒数です。
--imix	命令 MIX レポートを生成します。これは、IBS 設定と IBS イベントのプロファイリングのみでサポートされます。ネイティブ バイナリに対してのみサポートされます。
--ignore-system-module	システム モジュールからのサンプルを無視します。
--show-percentage	実際のサンプルではなく、サンプルの割合を表示します。
--show-sample-count	サンプル数を表示します。このオプションはデフォルトでオンです。
--show-event-count	イベントの発生回数を表示します。
--show-all-cachelines	キャッシュ解析のため、レポート セクションにすべてのキャッシュ ラインを表示します。デフォルトでは、複数のプロセス/スレッドがアクセスするキャッシュ ラインのみが表示されます。Windows および Linux プラットフォームでのメモリ設定レポートのみでサポートされます。
--bin-path <path>	バイナリ ファイルのパス。--bin-path は複数使用できます。
--src-path <path>	ソース ファイルのパス。--src-path は複数使用できます。

表 28. AMDuProfCLI の report コマンドのオプション

オプション	説明
--symbol-path <path1;...>	デバッグ シンボルのパス (セミコロン区切り)。--symbol-path は複数使用できます。
--report-output <path>	レポートをファイルに書き込みます。.csv 拡張子が含まれるパスは、ファイル パスであると見なされ、そのまま使用されます。.csv 拡張子がない場合、そのパスはディレクトリであると見なされ、ディレクトリ内にデフォルト名でレポート ファイルが生成されます。
--stdout	コンソールまたはターミナルにレポートを出力します。
--retranslate	異なる変換オプションで、収集されたデータ ファイルの再変換を実行します。
--remove-raw-files	生のデータ ファイルを削除して、ディスク容量を回復します。
--export-session	別のシステムで解析のために使用できるように、必要なセッション ファイルの圧縮アーカイブを作成します。
--log-path <path-to-log-dir>	ログ ファイルが作成される場所へのパスを指定します。このオプションを指定しない場合、デフォルトでログ ファイルが作成される場所は、AMDUPROF_LOGDIR 環境変数で設定されたパス、\$TEMP パス (Linux、FreeBSD)、または %TEMP% パス (Windows) になります。 ログ ファイル名の形式は、\$USER-AMDuProfCLI.log (Linux、FreeBSD) または %USERNAME%-AMDuProfCLI.log (Windows) となります。
--enable-log	ログ ファイルへの追加ロギングを有効にします。
--enable-logts	ログ レコードのタイムスタンプを取り込みます。このオプションは、--enable-log option と一緒に使用する必要があります。

6.5.2 Windows 専用オプション

表 29. AMDuProfCLI の report コマンド – Windows 専用オプション

オプション	説明
--symbol-server <path1;...>	シンボル サーバーのディレクトリ (複数のパスをセミコロンで区切ります)。例: Microsoft のシンボル サーバー (https://msdl.microsoft.com/download/symbols)。--symbol-server は複数使用できます。
--symbol-cache-dir <path>	シンボル サーバーからダウンロードしたシンボル ファイルを保存するパス。
--legacy-symbol-downloader	Microsoft Symshr を使用してシンボルをダウンロードします。デフォルトでは、AMD シンボル ダウンローダーが使用されます。

6.5.3 Linux 専用オプション

表 30. AMDuProfCLI の report コマンド – Linux 専用オプション

オプション	説明
<code>--host <hostname></code>	このオプションは、<code>--input-dir</code> オプションと一緒に使用します。指定されたホストに属するレポートを生成します。サポートされるオプションは次のとおりです。 • <code><hostname></code>: 特定のホストに属するプロセスをレポートします。 • <code>all</code>: すべてのプロセスをレポートします。 注記: <code>--host</code> を使用しない場合、そこからレポートを生成するシステムに属するプロセスのみがレポート対象になります。システムがクラスター内のマスター ノードである場合、そのクラスター内の辞書順で最初のホストに対してレポートが生成されます。
<code>--category <PROFILE></code>	特定のプロファイリング カテゴリのレポートのみを生成します。複数のカテゴリをカンマ区切りで指定できます。このオプションを使用しない場合、すべてのカテゴリに対してレポートが生成されます。<code>--category</code> を複数使用できます。サポートされるカテゴリは次のとおりです。 • <code>cpu</code> – CPU プロファイリング専用のレポートを生成します。 • <code>mpi</code> – MPI トレース専用のレポートを生成します。 • <code>openmp</code> – OpenMP トレース専用のレポートを生成します。 • <code>trace</code> – トレース イベント専用のレポートを生成します。 • <code>gputrace</code> – GPU トレース専用のレポートを生成します。 • <code>gpuprof</code> – GPU プロファイリング専用のレポートを生成します。 例: <pre>--category cpu,mpi,trace,gputrace,gpuprof --category mpi --category cpu --category trace --category gputrace --category gpuprof</pre>
<code>--funccount-interval <funccount-interval></code>	関数カウント詳細レポートを出力するための時間間隔を秒数で指定します。このオプションが指定されない場合、プロファイル時間全体の関数カウントが生成されます。

6.5.4 例

Windows

- 生データ ファイルからレポートを生成します。

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

- 生データ ファイルから IMIX レポートを生成します。

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

- 生データ ファイルから、pmc イベントで並べ替えたレポートを生成します。

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

- 生データ ファイルから、ibs-op イベントで並べ替えたレポートを生成します。

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

- 消費電力サンプルの生データ ファイルからレポートを生成します

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

- シンボル サーバーのパスを指定してレポートを生成します。

```
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>
```

- いずれかの事前定義ビューに関するレポートを生データ ファイルから生成します。

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

- ソース パスとバイナリ パスを指定して、生データ ファイルからレポートを生成します。

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

Linux

- 生データ ファイルからレポートを生成します。

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

- 生データ ファイルから IMIX レポートを生成します。

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

- 生データ ファイルから、pmc イベントで並べ替えたレポートを生成します。

```
$ AMDuProfCLI report -s event=pmcxc0,user=1,os=0 -i /tmp/cpuprof-ebp/<SESSION-DIR>
```

- 生データ ファイルから、ibs-op イベントで並べ替えたレポートを生成します。

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

- 生データ ファイルからトレース レポートを生成します。

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

- 生データ ファイルから GPU トレース レポートを生成します。

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

- 生データ ファイルから GPU プロファイル レポートを生成します。

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

6.6 Translate コマンド

translate コマンドは、生のプロファイル データを処理して、データベース ファイル内にサンプル情報を生成します。これらのデータベースを GUI または CLI にインポートして、レポートの生成に使用できます。

構造:

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

一般的な使用法:

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

6.6.1 オプション

次の表に、AMDuProfCLI の translate コマンドのオプション一覧を示します。

表 31. AMDuProfCLI Translate コマンドのオプション

オプション	説明
-h --help	ヘルプ情報を表示します。
-i --input-dir <directory-path>	収集されたデータを含むディレクトリへのパス。
--time-filter <T1:T2>	処理時間を T1 と T2 の間に限定します。ここで、T1 と T2 はプロファイル開始時間からの秒数です。
--agg-interval <low medium high INTERVAL>	このオプションを使用して、サンプル集計の間隔を設定します。これは、セッションを GUI にインポートするときに便利です。 low レベルの集計間隔を指定すると、GUI のタイムライン ビューの品質は高くなりますが、データベース サイズが大きくなります。 INTERVAL には集計間隔の数値をミリ秒で指定できます。
--bin-path <path>	バイナリ ファイルのパス。--bin-path は複数使用できます。
--symbol-path <path>	デバッグ シンボルのパス。--symbol-path を複数指定できます。
--inline	C および C++ 実行ファイルのインライン関数を抽出します。 注記: 1. このオプションは Windows ではサポートされていません。 2. このオプションを使用すると、レポート生成にかかる時間が長くなります。
--retranslate	異なる変換オプションで、収集されたデータ ファイルを再変換します。
--log-path <path-to- log-dir>	ログ ファイルが作成される場所へのパスを指定します。このオプションを指定しない場合、デフォルトでログ ファイルが作成される場所は、AMDUPROF_LOGDIR 環境変数で設定されたパスか、%TEMP% パスになります。 ログ ファイル名の形式は、\$USER-AMDuProfCLI.log (Linux、FreeBSD) または %USERNAME%-AMDuProfCLI.log (Windows) となります。
--enable-log	ログ ファイルへの追加ロギングを有効にします。
--enable-logts	ログ レコードのタイムスタンプを取り込みます。このオプションは、--enable-log オプションと一緒に使用する必要があります。
--remove-raw-files	生のデータ ファイルを削除して、ディスク容量を回復します。
--export-session	別のシステムで解析のために使用できるように、必要なセッション ファイルの圧縮アーカイブを作成します。

6.6.2 Windows 専用オプション

次の表に、translate コマンドの Windows 専用オプション一覧を示します。

表 32. Translate コマンド – Windows 専用オプション

オプション	説明
--symbol-server <path1;...>	シンボル サーバーへのリンク。例: Microsoft のシンボル サーバー (https://msdl.microsoft.com/download/symbols)。--symbol- サーバーを複数指定できます。
--symbol-cache-dir <path>	シンボル サーバーからダウンロードしたシンボルを保存するパス。
--legacy-symbol-downloader	Microsoft Symshr を使用してシンボルをダウンロードします。デフォルトでは、AMD シンボル ダウンローダーが使用されます。

6.6.3 Linux 専用オプション

次の表に、translate コマンドの Linux 専用オプション一覧を示します。

表 33. Translate コマンド – Linux 専用オプション

オプション	説明
--category <PROFILE>	指定されたプロファイリング カテゴリのみを処理します。複数のカテゴリをカンマ区切りで指定できます。このオプションが使用されない場合、すべてのカテゴリの生データファイルが処理されます。--category を複数指定できます。サポートされるカテゴリは次のとおりです。 - cpu - CPU プロファイリング - mpi - MPI トレース - openmp - OpenMP トレース専用のレポートを生成します。 - trace - ユーザー モードのトレース - gputrace - GPU トレース - gpuprof - GPU プロファイリング 例: <pre>--category cpu,mpi,trace,gputrace,gpuprof --category mpi --category cpu --category trace --category gputrace --category gpuprof</pre>
--host <hostname>	このオプションは、--input-dir オプションと一緒に使用します。指定されたホストに属するサンプルを処理します。サポートされるオプションは次のとおりです。 <hostname>: 特定のホストに属するプロセスのみを変換します。 all: すべてのプロセスを変換します。 注記: --host を使用しない場合、現在のシステムに属するプロセスのみが変換されます。システムがクラスター内のマスター ノードである場合、そのクラスター内の辞書順で最初のホストに対して処理が実行されます。
--kallsyms-path <path>	kallsyms 情報を含むファイルへのパス。パスを指定しない場合、デフォルトとして /proc/kallsyms が使用されます。
--vmlinux-path <path>	Linux カーネルのデバッグ情報ファイルへのパス。パスを指定しない場合、デフォルトのダウンロード パスからデバッグ情報ファイルが検索されます。

6.6.4 例

Windows

- すべての生データ ファイルを処理します。

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

- シンボル サーバーのパスを指定して、生データ ファイルを処理します。

```
> 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>
```

- ソース パスとバイナリ パスを指定して、生データ ファイルを処理します。

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

Linux

- すべての生データ ファイルを処理します。

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

- トレースの生データ ファイルを処理します。

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

- GPU トレースの生データ ファイルを処理します。

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

6.7 Timechart コマンド

timechart コマンドは、消費電力、熱、周波数メトリクスなどのシステム特性を収集およびレポートし、テキストまたは CSV レポートを生成します。

注記: timechart コマンドがサポートされるのは、Windows と Linux のみです。

構造:

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

<PROGRAM> — 消費電力メトリクスの収集を開始する前に起動するアプリケーションを指定します。

<ARGS> — 起動アプリケーションに渡す引数のリストを指定します。

一般的な使用法:

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

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

6.7.1 オプション

表 34. AMDuProfCLI Timechart コマンドのオプション

オプション	説明
-h --help	ヘルプ情報を表示します。
--list	サポートされるデバイスとカテゴリをすべて表示します。
-e --event <type...>	指定された組み合わせのデバイス タイプおよび/またはカテゴリ タイプに対して、カウンターを収集します。 サポートされるデバイスとカテゴリの一覧を表示するには、timechart --list コマンドを使用します。 注記: -e は複数指定できます。
-t --interval <n>	サンプリング間隔 n をミリ秒で指定します。最小値は 10 ms です。
-d --duration <n>	プロファイル時間 n を秒で指定します。
--affinity <core...>	コアのアフィニティ。コア ID のカンマ区切りリスト。コア ID の範囲も指定できます。例: 0-3。デフォルト アフィニティは、使用可能なすべてのコアです。 アフィニティは、起動アプリケーションに対して設定されます。
-w --working-dir <dir>	ターゲットの起動アプリケーションの作業ディレクトリを設定します。
-f --format <fmt>	出力ファイルの形式。サポートされる形式は次のとおりです。 • txt: テキスト (.txt) 形式。 • csv: カンマ区切り値 (.csv) 形式。 デフォルトのファイル形式は .csv です。
-o --output-dir <dir>	出力ディレクトリのパス。

6.7.2 例

Windows

- 100 ミリ秒のサンプリング間隔で 10 秒間にわたり、すべての消費電力カウンター値を収集します。

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

- 10 秒間にわたりすべての周波数カウンター値を収集し、500 ミリ秒ごとにサンプリングし、結果を csv ファイルにダンプします。

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

- 10 秒間にわたり、コア 0 ~ 3 のすべての周波数カウンター値を収集し、500 ミリ秒ごとにサンプリングし、結果をテキスト ファイルにダンプします。

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

Linux

- 100 ミリ秒のサンプリング間隔で 10 秒間にわたり、すべての消費電力カウンター値を収集します。

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

- 10 秒間にわたりすべての周波数カウンター値を収集し、500 ミリ秒ごとにサンプリングし、結果を csv ファイルにダンプします。

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

- 10 秒間にわたり、コア 0 ~ 3 のすべての周波数カウンター値を収集し、500 ミリ秒ごとにサンプリングし、結果をテキスト ファイルにダンプします。

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

6.8 Diff コマンド

diff コマンドでは、手作業によるイベント比較が自動化されるため、複数のプロファイルレポートの比較プロセスが効率化されます。生プロファイルデータ、処理済みファイル、またはデータベース ファイルを処理し、収集されたプロファイルに対するマークダウン比較レポートを生成します。生成されたマークダウンファイルには詳細な関数データが含まれ、比較されたプロファイルに対する包括的なインサイトが提供されます。

また、diff コマンドを使用して、ベース プロファイルパスのみを指定すると、単一のプロファイルレポートを生成できます。これにより、個別レポートの生成が簡素化され、利便性と効率が高くなります。

プロファイルの比較中は常に、1 つのベース プロファイルと複数の非ベース プロファイルがあります。有効な比較結果は、ベース プロファイルと非ベース プロファイルの両方に存在する関数に対してのみ獲得できます。

デフォルトでは、比較結果はソース ビューに表示されます。ソース ビュー テーブルでは、ファイル、行、ソース コード、アドレス、命令、コード バイト、イベントなどの情報が関数別に表示されます。この包括的なビューにより、比較したプロファイルを詳細に解析できるようになります。

注記: 意味のある正確な比較結果を取得するには、ベース プロファイルと非ベース プロファイルで、対応する関数が比較に使用できるようになっていることが重要です。

構造:

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

一般的な使用法:

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

6.8.1 プロファイル比較の適格基準

正確で意味のあるプロファイル比較を実現するには、次の条件が満たされている必要があります。

- 同じイベント: 比較対象のプロファイルで、同じイベントが収集されている必要があります。これにより、同等の関連性があるデータに関する比較を実行できます。
- 同じプロファイル時間 (指定されている場合): 時間 (-d) オプションが指定されている場合、比較されている複数のプロファイルの時間が一致している必要があります。これにより、プロファイルでカバーされる時間の一貫性が確保されます。
- システム全体のプロファイルではない: システム全体のプロファイルは直接比較できません。このため、個別のプロセスまたはスレッド レベルのプロファイルのみが比較対象となります。
- 同じプロファイル データ制限 (使用されている場合): プロファイリング中に --limit-size または --limit-data オプションが使用されている場合、比較対象のプロファイルのデータ制限が一致する必要があります。これにより、収集されたプロファイル データ サイズの一貫性が確保されます。
- 同じインライン関数プロファイリング (--inline): インライン関数のプロファイリングで --inline オプションが使用されている場合、比較対象のプロファイルでは、同じインライン関数のプロファイリング設定が使用されている必要があります。これにより、比較時のインライン関数の処理に一貫性が確保されます。

6.8.2 オプション

次の表に、diff コマンドのオプション一覧を示します。

表 35. AMDuProfCLI diff コマンドのオプション

オプション	説明
-h --help	このヘルプ情報をコンソール/ターミナルに表示します。
--baseline <directory-path>	収集されたデータを含むディレクトリへのパス。このディレクトリ内のプロファイルデータは、ベース プロファイルとして扱われ、その他すべてのプロファイルを比較する基準となります。
--with <directory-path>	収集されたデータを含むディレクトリへのパス。--with が指定されたプロファイルは非ベース プロファイルと見なされ、ベース プロファイルと比較されます。--with を複数使用すると、比較するための非ベース プロファイルを複数指定できます。
-i, --input-dir <directory-path>	収集されたデータを含むディレクトリへのパス。-i は複数指定できます。最初に指定した -i はベース セッションと見なされ、その後に指定したすべての -i は、非ベース セッションと見なされます。 注記: -i, --input-dir を使用する場合、--baseline または --with オプションと同時に使用しないでください。--baseline と -i を一緒に使用すると、--baseline オプションが優先されて、ベース セッションと見なされます。--baseline オプションが指定されない場合、最初に指定した -i が自動的にベース セッションと見なされます。
--output-dir -o <directory-path>	マークダウン比較レポートが生成される場所のパス。

表 35. AMDuProfCLI diff コマンドのオプション (続き)

オプション	説明
<code>--type <comparison-type></code>	実行する比較のタイプを指定します。サポートされる比較タイプは次のとおりです。 • name: このタイプを指定すると、ベース プロファイル内の上位「n」個の関数のみが、非ベース プロファイル内の対応する関数と比較されます。比較のフォーカスは、プロファイル間で類似した関数に置かれます。 • order: このタイプを指定すると、すべてのプロファイル内の上位「n」個の関数が、プロファイル順に表示されます。表示順は次のとおりです。最初にベース プロファイル、次に 1 番目の非ベース プロファイル、2 番目の非ベース プロファイルのように続きます。ベース プロファイルに含まれる関数との比較が実行され、プロファイル間で類似した関数のみが対象となります。 デフォルトの比較タイプは、name です。
<code>--alias <base-fun,non-base-fun,... base-fun-1,non-base-fun-1,... ...></code>	非ベース プロファイル内の関数名が変更された場合に、ベース プロファイル内の対応する関数名と比較される非ベース プロファイル内の関数名を指定します。 複数の関数を指定するには、区切り文字としてパイプ記号「 」を使用します。それぞれの関数セットに対して、ベース プロファイルと非ベース プロファイルの間で関数名を区切るにはカンマを使用します。
<code>--show-percentage</code>	比較結果は割合 (%) で表示されます。
<code>--cutoff <n></code>	レポートする関数の数を制限するためのカットオフ。「n」は、各種レポート セクションに記載されるエントリの最大数です。デフォルト値は 10 です。
<code>--sort-by -s <EVENT></code>	レポート対象プロファイル データを並べ替える基準として、タイマー、PMC、または IBS イベントを、キー=値ペアのカンマ区切り形式で引数として指定します。サポートされているキーは次のとおりです。 • <code>event=<timer ibs-fetch ibs-op pmcxNNN></code>。ここで、NNN は 16 進数のコア PMC イベント ID です。 • <code>umask=<unit-mask></code> • <code>user=<0 1></code> • <code>os=<0 1></code> サポートされる PMC イベントの一覧を表示するには、<code>info --list pmu-events</code> コマンドを使用します。引数の詳細は次のとおりです。 • <code>umask</code> — 10 進数または 16 進数のユニット マスク。PMC イベントのみに適用されます。 • <code>user</code>、<code>os</code> — ユーザー モードと OS モード。PMC イベントのみに適用されます。 <code>--sort-by (-s)</code> は複数指定できません。
<code>--view <view-config></code>	指定されたビュー ファイルに含まれるイベントのみを比較します。サポートされる <code>view-configs</code> のリストを表示するには、 <code>info --list view-configs</code> コマンドを使用します。
<code>--stdout</code>	比較レポートはファイルに保存されるだけでなく、ターミナルまたはコマンド ライン インターフェイスに表示されます。
<code>--src-path <path1;...></code>	ベース プロファイルのソース ファイル ディレクトリ (複数のパスをセミコロンで区切ります)。対応するファイル ディレクトリが個別に指定されていない場合、これは非ベース プロファイル向けと見なされます。 <code>--src-path</code> は複数使用できます。

表 35. AMDuProfCLI diff コマンドのオプション (続き)

オプション	説明
--bin-path <path1;...>	ベース プロファイルのバイナリ ファイルのパス。対応する bin パスが個別に指定されていない場合、これは非ベース プロファイル向けと見なされます。 --bin-path は複数使用できます。
--src-path1 <path1;...>	1 番目の非ベース プロファイルのソース ファイル ディレクトリ (複数のパスをセミコロンで区切ります)。 --src-path1 は複数使用できます。
--bin-path1 <path1;...>	1 番目の非ベース プロファイルのバイナリ ファイル パス。 --bin-path1 は複数使用できます。
--src-path2 <path1;...>	2 番目の非ベース プロファイルのソース ファイル ディレクトリ (複数のパスをセミコロンで区切ります)。 --src-path2 は複数使用できます。
--bin-path2 <path1;...>	2 番目の非ベース プロファイルのバイナリ ファイル パス。 --bin-path2 は複数使用できます。
--src-path3 <path1;...>	3 番目の非ベース プロファイルのソース ファイル ディレクトリ (複数のパスをセミコロンで区切ります)。 --src-path3 は複数使用できます。
--bin-path3 <path1;...>	3 番目の非ベース プロファイルのバイナリ ファイル パス。 --bin-path3 は複数使用できます。

6.8.3 例

Windows

それぞれの目的に応じて次のコマンドを使用します。

- ベース プロファイル データと後続のプロファイル データとの比較レポートを生成します。

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

- i オプションを使用して比較レポートを生成します。

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

- セッション間で固有のエントリを無視せずに、比較レポートを生成します。

```
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
```

- ベース プロファイル データと後続のプロファイル データとの比較レポートを生成し、ibs-op イベントで並べ替えます。

```
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
```

- 差分を割合 (%) で示した比較レポートを生成します。

```
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
```

- ベース プロファイル データと、セッション間で関数名が変更された後続のプロファイル データとの比較レポートを生成します。

```
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
```

- ベース プロファイル データと複数の後続プロファイル データとの比較レポートを生成します。

```
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
```

- いずれかの事前定義ビューに対して比較レポートを生成します。

```
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
```

- ソース パスとバイナリ パスを指定して、比較レポートを生成します。

```
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

- ベース プロファイル データと後続のプロファイル データとの比較レポートを生成します。

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

- ベース プロファイル データと後続のプロファイル データとの比較レポートを生成し、PMC イベントで並べ替えます。

```
$ 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 コマンド

profile コマンドを実行すると、パフォーマンス プロファイル データが収集されて処理され、読みやすい形式でプロファイル レポートが生成されます。このコマンドは、collect コマンドと report コマンドの組み合わせの代わりに使用できます。

構造:

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

<PROGRAM>— プロファイリングの対象となる起動アプリケーションを指定。

<ARGS>— 起動アプリケーションに渡す引数のリストを指定。

一般的な使用法:

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

6.9.1 オプション

次の表に、profile コマンドのオプション一覧を示します。

表 36. AMDuProfCLI profile コマンドのオプション

オプション	説明
-h --help	ヘルプ情報をコンソール/ターミナルに表示します。
-o --output-dir <directory-path>	収集されたデータ ファイルが保存されるベースディレクトリのパス。このディレクトリ内に、新しいサブディレクトリが作成されます。
--config <config>	サンプル収集で使用する事前定義サンプリング設定。 サポートされる設定の一覧を表示するには、info --list collect-configs コマンドを使用します。--config は複数指定できます。

表 36. AMDuProfCLI profile コマンドのオプション (続き)

オプション	説明
-e --event or <predefined-event>	-e、--event では、引数が事前に定義された事前定義イベントを直接使用できます。 または、より詳細なパラメーターを指定する方法として、タイマー、PMU、IBS イベント、または事前定義イベントを、キー=値ペアのカンマ区切り形式で引数に指定できます。サポートされているキーは次のとおりです。 • event=<timer ibs-fetch ibs-op> または <PMU-event> または <predefined-event> • mask=<unit-mask> • user=<0 1> • os=<0 1> • cmask=<count-mask> (値の範囲は 0x0 から 0x7f) • inv=<0 1> • interval=<sampling-interval> • frequency=<frequency (n)> (コア PMC イベントのみでサポート。周波数は Hz で指定) • ibsop-count-control=<0 1> (ibs-op イベントの場合) • loadstore (ibs-op イベントの場合。Windows プラットフォームのみ) • ibsop-l3miss (ibs-op イベントの場合。AMD “Zen4” プロセッサのみでサポート) • ibsfetch-l3miss (ibs-fetch イベントの場合。AMD “Zen4” プロセッサのみでサポート) • call-graph 注記: 1. 事前定義イベントには umask を指定する必要はありません。 2. コールスタック サンプルの収集に関連する引数を指定するには、専用オプション --call-graph を使用します。 引数の詳細は、次のとおりです。 • user – ユーザー スペースのサンプル収集を有効 (1) または無効 (0) にします。 • os – カーネル スペースのサンプル収集を有効 (1) または無効 (0) にします。 • interval – サンプル収集の間隔。タイマーでは、時間間隔をミリ秒で指定します。PMU と事前定義イベントの場合は、イベント発生回数です。IBS フェッチの場合はフェッチ回数です。IBS オペレーションの場合、サイクル数またはディスパッチ数になります。 • op-count-control – IBS オペレーションのサンプリング数を、サイクル数 (0) またはディスパッチ数 (1) から選択します。 • loadstore – IBS オペレーションのロード/ストア サンプル収集だけを有効にします。その他の IBS オペレーション サンプルは収集されません。 • ibsop-l3miss – l3 ミス発生時のみ、IBS オペレーション サンプル収集を有効にします。 例: 「-e event=ibs-op,interval=100000,ibsop-l3miss」 • ibsfetch-l3miss – l3 ミス発生時のみ、IBS フェッチのサンプル収集を有効にします。 例: 「-e event=ibs-fetch,interval=100000,ibsfetch-l3miss」

表 36. AMDuProfCLI profile コマンドのオプション (続き)

オプション	説明
	引数が指定されない場合のデフォルト値は次のとおりです。 • umask=0 • cmask=0x0 • user=1 • os=1 • inv=0 • ibsop-count-control=0 (ibs-op イベントの場合) 必要に応じて次のコマンドを使用します。 • info --list predefined-events: サポートされる事前定義イベントの一覧を表示 • info --list pmu-events: サポートされる PMC イベントの一覧を表示 --event (-e) は複数指定できます。
-p --pid <PID...>	特定の実行中プロセスに接続することで、既存プロセスをプロファイリングします。プロセス ID の区切りにはカンマを使用します。 注記: 一度に最大で 512 のプロセスに接続できます。
-a --system-wide	システム全体のプロファイル (SWP) このフラグが設定されていない場合、コマンド ライン ツールでプロファイリングされるのは、起動したアプリケーションか、-p オプションで接続したプロセス ID のみになります。
-c --cpu <core...>	プロファイリングする CPU のカンマ区切りリスト。「-」を使用すると、CPU の範囲を指定できます。例: 0-3。 注記: Windows では、選択されたコアが 1 つのプロセッサ グループのみに属している必要があります。たとえば、0-63、64-127 などです。
-d --duration <n>	指定された「n」秒間だけ、プロファイリングを実行します。
--interval <num>	PMC イベントのサンプリング間隔。 注記: この間隔によって、個別イベントに指定されたサンプリング間隔がオーバーライドされます。
--affinity <core-id...>	プロファイル対象となる起動アプリケーションのコアのアフィニティを設定します。コア ID のカンマ区切りリスト。 コア ID の範囲を指定する必要があります。例: 0-3。 デフォルト アフィニティは、使用可能なすべてのコアです。
--no-inherit	起動アプリケーションの子 (プロファイル対象アプリケーションによって起動されるプロセス) はプロファイリングしません。
-b --terminate	プロファイルデータの収集が終了した後で、起動アプリケーションを終了します。中止するのは起動アプリケーションのプロセスのみです。子プロセスがある場合、その実行は継続される可能性があります。
--thread <thread=concurrency>	スレッド同時並行性
--start-delay <n>	n 秒の開始遅延。指定された時間の経過後にプロファイリングを開始します。「n」が 0 の場合は何も影響はありません。

表 36. AMDuProfCLI profile コマンドのオプション (続き)

オプション	説明
--start-paused	プロファイリングが無期限に一時停止します。ターゲット アプリケーションが、プロファイル制御 API を使用してプロファイリングを再開します。このオプションは、再開/一時停止 API (「 AMDProfileControl API 」参照) を使用したプロファイル データ収集の制御が、起動アプリケーションにインストールメント化されている場合のみ使用してください。
-w --working-dir <path>	作業ディレクトリを指定します。デフォルトは、現在の作業ディレクトリです。
--log-path <path-to-logdir>	ログ ファイルが作成される場所へのパスを指定します。このオプションを指定しない場合、デフォルトでログ ファイルが作成される場所は、AMDUPROF_LOGDIR 環境変数で設定されたパス、\$TEMP パス (Linux、FreeBSD)、または %TEMP% パス (Windows) になります。 ログ ファイル名の形式は、\$USER-AMDuProfCLI.log (Linux、FreeBSD) または %USERNAME%-AMDuProfCLI.log (Windows) となります。
--enable-log	ログ ファイルへの追加ロギングを有効にします。
--enable-logts	ログ レコードのタイムスタンプを取り込みます。--enable-log オプションと共に使用します。
--limit-size <n>	収集データのファイル サイズ (MB) が上限を超えた場合にプロファイリングを停止するには、このオプションを使用します。このオプションは、今後のリリースで廃止される予定です。
--frequency <n> --freq <n> -F <n>	コア PMC イベントに対して、指定された周波数「n」(Hz) でのデータ収集を有効にします。 注記: この周波数によって、個別イベントに指定されたサンプリング周波数がオーバーライドされます。
--detail	詳細なレポートを生成します。
--group-by <section>	生成されるレポートを指定します。サポートされるレポート オプションは次のとおりです。 • process: プロセスの詳細をレポート • module: モジュールの詳細をレポート • thread: スレッドの詳細をレポート このオプションが適用されるのは、--detail オプションと共に使用する場合のみです。デフォルトは、group-by process です。
--cutoff <n>	レポートするプロセス、スレッド、モジュール、関数の数を制限するためのカットオフ。「n」は、各種レポート セクションに記載されるエントリの最小数です。デフォルト値は 10 です。
--view <view-config>	指定されたビュー ファイルに含まれるイベントのみをレポートします。サポートされるビュー設定の一覧を表示するには、info --list view-configs コマンドを使用します。
--inline	C、C++ 実行ファイルのインライン関数を表示します。 注記: 1. このオプションは Windows ではサポートされていません。 2. このオプションを使用すると、レポート生成にかかる時間が長くなります。

表 36. AMDuProfCLI profile コマンドのオプション (続き)

オプション	説明
--show-sys-src	システム モジュール関数の詳細な関数レポートを、(デバッグ情報を使用できる場合は) ソース ステートメント付きで生成します。
--src-path <path1;...>	ソース ファイルのディレクトリ (複数のパスをセミコロンで区切ります)。--src-path は複数使用できます。
--disasm	詳細な関数レポートをアセンブリ命令付きで生成します。
--disasm-only	アセンブリ命令だけを含む関数レポートを生成します。
--disasm-style <att intel>	アセンブリ命令の構文を選択します。サポートされるオプションは、「att」または「intel」です。このオプションが指定されない場合、使用されるデフォルト スタイルは「intel」です。
-s --sort-by <EVENT>	レポート対象プロファイル データを並べ替える基準として、タイマー、PMC、または IBS イベントを、キー=値ペアのカンマ区切り形式で引数として指定します。 サポートされているキーは次のとおりです。 - event=<timer ibs-fetch ibs-op pmcxNNN>、ここで NNN は 16 進数のコア PMC イベント ID。 - umask=<unit-mask> - cmask=<count-mask> - inv=<0 1> - user=<0 1> - os=<0 1> サポートされる PMC イベントの一覧を表示するには、info --list pmu-events コマンドを使用します。 引数の詳細は、次のとおりです。 - umask — 10 進数または 16 進数のユニット マスク。PMC イベントのみに適用されます。 - cmask — 10 進数または 16 進数のカウント マスク。PMC イベントのみに適用されます。 - user、os — ユーザー モードと OS モード。PMC イベントのみに適用されます。 - inv — 逆カウント マスク。PMC イベントのみに適用されます。 -sort-by (-s) を複数指定することはできません。
--agg-interval <low medium high INTERVAL>	このオプションを使用して、サンプル集計の間隔を設定します。これは、セッションを GUI にインポートするときに便利です。 「low」レベルの集計間隔を指定すると、GUI のタイムライン ビューの品質は高くなりますが、データベース サイズが大きくなります。 INTERVAL には集計間隔の数値をミリ秒で指定できます。
--time-filter <T1:T2>	レポート生成時間を T1 と T2 の間に限定します。ここで、T1 と T2 はプロファイル開始時間からの秒数です。
--imix	命令 MIX レポートを生成します。IBS 設定、IBS イベント プロファイリング、ネイティブ バイナリのみでサポートされます。
--ignore-system-module	システム モジュールからのサンプルを無視します。
--show-percentage	実際のサンプルではなく、サンプルの割合を表示します。

表 36. AMDuProfCLI profile コマンドのオプション (続き)

オプション	説明
--show-sample-count	サンプル数を表示します。このオプションはデフォルトでオンです。
--show-event-count	イベントの発生回数を表示します。
--show-all-cachelines	キャッシュ解析のため、レポート セクションにすべてのキャッシュ ラインを表示します。デフォルトでは、複数のプロセス/スレッドがアクセスするキャッシュ ラインのみが表示されます。 Windows および Linux プラットフォームでのメモリ設定レポートのみでサポートされます。
--bin-path <path>	バイナリ ファイルのパス。--bin-path は複数使用できます。
--src-path <path>	ソース ファイルのパス。--src-path は複数使用できます。
--symbol-path <path1;...>	デバッグ シンボルのパス (セミコロン区切り)。--symbol-path は複数使用できます。
--report-output <path>	レポートをファイルに書き込みます。.csv 拡張子が含まれるパスは、ファイルパスであると見なされ、そのまま使用されます。.csv 拡張子が使用されていない場合、そのパスはディレクトリであると見なされ、ディレクトリ内にデフォルト名でレポート ファイルが生成されます。
--stdout	コンソールまたはターミナルにレポートを出力します。
--retranslate	異なる変換オプションで、収集されたデータ ファイルの再変換を実行します。
--ascii event-dump	IBS オペレーションのプロファイル サンプルの ASCII ダンプを生成するには、このオプションを使用します。 注記: このオプションを指定すると、変換に遅延が生じる場合があります。
--no-report	収集と変換のみを実行するには、このオプションを使用します。
--remove-raw-files	生のデータ ファイルを削除して、ディスク容量を回復します。
--export-session	別のシステムで解析のために使用できるように、必要なセッション ファイルの圧縮アーカイブを作成するには、このオプションを使用します。

6.9.2 Windows 専用オプション

次の表に Windows 専用のプロファイル コマンド一覧を示します。

表 37. AMDuProfCLI profile コマンドの Windows オプション

オプション	説明
--call-graph <I:D:S:F>	コールスタック サンプリングを有効にします。アンワインド間隔 (I) をミリ秒で指定し、アンワインドの深さ (D) の値を指定します。スコープ (S) を指定するには、次のいずれかを選択します。 • user: ユーザー スペースのコードのみを収集対象とします。 • kernel: カーネル スペースのコードのみを収集対象とします。 • all: ユーザー空間とカーネル空間で実行されるコードを収集対象とします。 コンパイラによるフレーム ポインターの省略 (F) が原因で見つからないフレームの収集方法を指定します。 • fpo: フレーム ポインターが使用できない場合、アンワインド情報を使用してコールスタック情報を収集します。 • fp: フレームポインターを使用して、コールスタック情報を収集します。
-g	次を指定するのと同じです。--call-graph 1:128:user:fp
--thread <thread=concurrency>	ランタイム スレッドの詳細を収集します。
-m --data-buffer-count <size>	ドライバーによるデータ収集に使用されるバッファのサイズ (コアあたりのページ数) を指定します。デフォルト サイズは、コアあたり 512 ページです。
--trace os	ターゲットのドメイン OS をトレースします。サポートされるのは、「スケジュール イベント」のみです。OS トレース イベントの一覧を表示するには、「info --list ostrace-events」コマンドを使用します。
--symbol-server <path1;...>	シンボル サーバーのディレクトリ (複数のパスをセミコロンで区切ります)。 例: Microsoft のシンボル サーバー (https://msdl.microsoft.com/download/symbols)。 --symbol-server は複数使用できます。
--symbol-cache-dir <path>	シンボル サーバーからダウンロードしたシンボル ファイルを保存するパス。
--legacy-symbol-downloader	Microsoft Symshr を使用してシンボルをダウンロードするには、このオプションを使用します。デフォルトで、シンボルのダウンロードには AMD シンボル ダウンローダーが使用されます。
--limit-data <n>	収集データのファイル サイズ (MB) が上限を超えた場合にプロファイリングを停止するには、このオプションを使用します。「--overwrite」オプションと一緒に使用された場合、上限は収集が停止される前になります。サイズは、メガバイト (M/m)、ギガバイト (G/g)、または秒 (secs) を末尾に付けて指定できます。
--overwrite	プロファイル データ収集モードをリング バッファとして設定します。収集の上限は、--limit-data オプションを使用して設定できます。デフォルトの --limit-data では、生のデータ ファイル サイズがコアあたり 512 ページに制限されます。

6.9.3 Linux 専用オプション

次の表に、Linux 専用コマンドの一覧を示します。

表 38. AMDuProfCLI profile コマンドの Linux オプション

オプション	説明
--call-graph <F:N>	コールスタック サンプリングを有効にします。コンパイラによるフレーム ポインターの省略が原因で見つからないフレームを収集するか、無視するかを指定 (F) します。 • fpo dwarf: サンプル収集中にプロセス コールスタックを収集し、DWARF 情報を使用してコールスタックを再構築します。 • fp: フレームポインターを使用して、コールスタック情報を収集します。 F = fpo の場合、「N」によって、1 回のサンプル収集で収集する最大スタック サイズがバイトで指定されます。スタック サイズの有効な範囲は、16 ~ 32768 です。「N」が 8 の倍数でない場合、最も近い 8 の倍数まで引き下げられます。デフォルト値は 1024 バイトです。 注記: 大きい N 値を指定すると、生成される生データ ファイルが非常に大きくなります。 F = fp の場合、「N」の値は無視されるため、指定する必要はありません。
-g	次を指定するのと同じです。--call-graph fp
--tid <TID,...>	特定の実行中スレッドに接続することで、既存スレッドをプロファイリングします。スレッド ID の区切りにはカンマを使用します。
--trace <TARGET>	ターゲット ドメインをトレースします。TARGET には、次に示すオプションから 1 つ以上を指定します。 mpi[=<openmpi mpich>,<lwt full>] 次のとおりに MPI インプリメンテーション タイプを指定します。 OpenMPI ライブラリをトレースする場合、「openmpi」 Intel MPI など、MPICH とその派生ライブラリをトレースする場合、「mpich」 次のとおりにトレースのスコープを指定します。 軽量トレースの場合、「lwt」 完全なトレースの場合、「full」 「--trace mpi」はデフォルトで「--trace mpi=mpich,full」となります。 • openmp - OpenMP アプリケーションをトレースする場合。これは、--omp オプションと同じです。 • os[=<event1,event2,...>] — イベント名とオプションのしきい値をカンマ区切りリストで指定します。syscall イベントと memtrace イベントは、オプションのしきい値を <event:threshold> として受け取ります。OS トレース イベントの一覧を表示するには、info --list ostrace-events コマンドを使用します。 • user=<event1,event2,...> — イベント名としきい値をカンマ区切りリストで指定します。これらのイベントはユーザー モードで収集されます。ユーザー モードでサポートされるトレース イベントの一覧を表示するには、info --list trace-events コマンドを使用します。 • gpu[=<hip,hsa>] — GPU トレースのドメインを指定します。デフォルトでは、ドメインは「hip,hsa」に設定されています。

表 38. AMDuProfCLI profile コマンドの Linux オプション (続き)

オプション	説明
--buffer-size <size>	OS トレース バッファに割り当てるページ数。デフォルト値は、コアあたり 256 ページです。 トレース データの損失を少なくするには、ページ数を増やします。このオプションを使用できるのは、OS トレース (--trace os) の場合のみです。
--max-threads <thread-count>	OS トレースの対象とする最大スレッド数。起動アプリケーションの場合、デフォルト値は 1024 です。システム全体のトレース (-a オプション) の場合は 32768 です。 アプリケーションのスレッド数がデフォルト制限を超えた場合は、この制限値を増やします。そうしない場合、動作が不定になります。 - 起動アプリケーション - 有効な範囲: 1 ~ 4096 - システム全体 - 有効な範囲: 1 ~ 4194304
--func <module:function-pattern>	トレースする関数を、ライブラリ、実行ファイル、またはカーネルから指定します。 function-pattern には、関数名か、「*」で終わる部分的な名前を指定できます。 「*」のみを指定すると、モジュールに含まれるすべての関数がトレースされます。 module にはライブラリまたは実行ファイルを指定します。カーネル関数をトレースする場合は、module を「kernel」で置換します。 注記: モジュールの絶対パス/フルパスを指定することを推奨します。それ以外の場合は、現在の作業ディレクトリではなく、デフォルト ライブラリ パスに対して検索が実行されます。
--exclude-func <module:function-pattern>	除外する関数を、ライブラリ、実行ファイル、またはカーネルから指定します。 - function-pattern には、関数名か、「*」で終わる部分的な名前を指定できます。 「*」のみを指定すると、モジュールに含まれるすべての関数がトレースされます。 - module にはライブラリまたは実行ファイルを指定します。カーネル関数をトレースする場合は、module を「kernel」で置換します。 注記: モジュールの絶対パスを指定することを推奨します。それ以外の場合は、現在の作業ディレクトリではなく、デフォルト ライブラリ パスに対して検索が実行されます。
-m --mmap-pages <size>	カーネル メモリ マップド データ バッファのサイズを設定します。サイズは、ページ数で指定するか、バイト (B/b)、キロバイト (K/k)、メガバイト (M/m)、ギガバイト (G/g) を末尾に付けて指定できます。
--mpi	MPI アプリケーションの CPU プロファイリング データを収集する際、このオプションを指定します。MPI トレースの場合、--trace オプションを参照してください。
--kvm-guest <pid>	guest サイドのパフォーマンス プロファイルを収集するために、プロファイリングする qemu-kvm プロセスの PID を指定します。
--guest-kallsyms <path>	ローカル ホストにコピーされた guest/proc/kallsyms のパスを指定します。AMD uProf はこれを読み取って、ゲスト カーネル シンボルを取得します。
--guest-modules <path>	ローカル ホストにコピーされた guest/proc/modules のパスを指定します。AMD uProf はこれを読み取って、ゲスト カーネルのモジュール情報を取得します。
--guest-search-path <path>	ローカル ホストにコピーされたゲスト vmlinux とカーネル ソースのパスを指定します。AMD uProf はこれを読み取って、ゲスト カーネルのモジュール情報を解決します。

表 38. AMDuProfCLI profile コマンドの Linux オプション (続き)

オプション	説明
<code>--host <hostname></code>	このオプションは、<code>--input-dir</code> オプションと一緒に使用します。指定されたホストに属するレポートを生成します。サポートされるオプションは次のとおりです。 • <code><hostname></code>: 特定のホストに属するプロセスをレポートします。 • <code>all</code>: すべてのプロセスをレポートします。 注記: <code>--host</code> を使用しない場合、レポートを生成するシステムに属するプロセスのみがレポート対象になります。システムがクラスター内のマスター ノードである場合、そのクラスター内の辞書順で最初のホストに対してレポートが生成されます。
<code>--category <PROFILE></code>	特定のプロファイリング カテゴリのレポートのみを生成します。複数のカテゴリをカンマ区切りで指定できます。このオプションを使用しない場合、すべてのカテゴリに対してレポートが生成されます。<code>--category</code> は複数指定できます。 サポートされるカテゴリは次のとおりです。 • <code>cpu</code>: CPU プロファイリング専用のレポートを生成します。 • <code>mpi</code>: MPI トレース専用のレポートを生成します。 • <code>openmp</code>: OpenMP トレース専用のレポートを生成します。 • <code>trace</code>: トレース イベント専用のレポートを生成します。[os] は廃止予定 • <code>gputrace</code>: GPU トレース専用のレポートを生成します。 • <code>gpuprof</code>: GPU プロファイリング専用のレポートを生成します。 例: <pre>--category cpu,mpi,trace,gputrace,gpuprof --category mpi --category cpu --category trace --category gputrace --category gpuprof</pre>
<code>--funccount-interval <funccount-interval></code>	関数カウント詳細レポートを出力するための時間間隔を秒数で指定します。このオプションが指定されない場合、プロファイル時間全体の関数カウントが生成されます。
<code>--branch-filter</code>	LBR データを取り込むには、このオプションを使用します。分岐フィルター タイプを指定します。 • <code>u</code>: ユーザー分岐 • <code>k</code>: カーネル分岐 • <code>any</code>: 任意の分岐タイプ • <code>any_call</code>: 任意の呼び出し分岐 • <code>any_ret</code>: 任意のリターン分岐 • <code>ind_call</code>: 間接呼び出し • <code>ind_jump</code>: 間接ジャンプ • <code>cond</code>: 条件付き分岐 • <code>call</code>: 直接呼び出し 上記のフィルターが設定されていない場合、デフォルトのフィルター タイプは「any」になります。 注記: 1. 上記のフィルターが設定されていない場合、デフォルトのフィルター タイプは「any」になります。 2. このオプションは、PMC イベントのみで正しく動作します。 3. これはプロセスごとに適用され、プロセスプロファイリングに関連付けられます。ただし、Java アプリケーションのプロファイリングには適用できません。

表 38. AMDuProfCLI profile コマンドの Linux オプション (続き)

オプション	説明
--vmlinux-path <path>	Linux カーネルのデバッグ情報ファイルへのパス。パスを指定しない場合、デフォルトのダウンロード パスからデバッグ情報ファイルが検索されます。

6.9.4 例

Windows

- アプリケーション *AMDTClassicMatMul.exe* を起動して、CYCLES_NOT_IN_HALT および RETIRED_INST イベントのサンプルを収集し、レポートを生成します。

```
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
```

- AMDTClassicMatMul.exe* アプリケーションを起動して、IBS サンプルを収集し、IMIX レポートを生成します。

```
AMDuProfCLI.exe profile --config ibs --imix -o c:\Temp\cpuprof-tbp AMDTClassicMatMul.exe
```

- AMDTClassicMatMul.exe* を起動して、Assess Performance のプロファイリングを 10 秒間実行し、レポートを生成します。

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

- AMDTClassicMatMul.exe* を起動して、SWP モードで IBS サンプルを収集し、ibs-op イベントで並べ替えたレポートを生成します。

```
C:\> AMDuProfCLI.exe profile --config ibs -a -s event=ibs-op -o c:\Temp\cpuprof-ibs-swp
AMDTClassicMatMul.exe
```

- SWP モードで 10 秒間にわたって TBP サンプルを収集し、レポートを生成します。

```
C:\> AMDuProfCLI.exe profile -a -o c:\Temp\cpuprof-tbp-swp -d 10
```

- AMDTClassicMatMul.exe* を起動して、コールスタック サンプリング付きで TBP を収集し、レポートを生成します。

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

- AMDTClassicMatMul.exe* を起動して、コールスタック サンプリング付きで TBP を収集し (アンワインド FPO 最適化スタック)、レポートを生成します。

```
C:\> AMDuProfCLI.exe profile --config tbp --call-graph 1:64:user:fpo -o c:\Temp\cpuprof-tbp
AMDTClassicMatMul.exe
```

- AMDTClassicMatMul.exe* を起動して、PMCx076 および PMCx0C0 のサンプルを収集し、pmcxc0 イベントで並べ替えたレポートを生成します。

```
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
```

- *AMDTClassicMatMul.exe* を起動して、50000 の間隔で IBS オペレーションのサンプルを収集し、ibs-op イベントで並べ替えたレポートを生成します。

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

- *AMDTClassicMatMul.exe* を起動して、スレッドの同時並行性および名前に関する TBP サンプルプロファイリングを実行し、レポートを生成します。

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

- *AMDTClassicMatMul.exe* を起動して、SWP モードで消費電力サンプルを収集し、レポートを生成します。

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

- PMCx076 および PMCx0C0 のサンプルを収集しますが、コールグラフ情報は PMCx0C0 のみに対して収集し、レポートを生成します。

```
C:\> AMDuProfCLI.exe profile -e event=pmcx76,interval=250000 -e
event=pmcxc0,interval=250000,call-graph -o c:\Temp\cpuprof-pmc AMDTClassicMatMul-bin
```

- *AMDTClassicMatMul.exe* を起動して、事前定義イベント RETIRED_INST と L1_DC_REFILLS.ALL のサンプルを収集し、レポートを生成します。

```
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
```

- *AMDTClassicMatMul.exe* を起動します。TBP、Assess Performance サンプルを収集し、レポートを生成します。

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

Linux

- *AMDTClassicMatMul.bin* アプリケーションを起動します。CYCLES_NOT_IN_HALT および RETIRED_INST イベントのサンプルを収集し、レポートを生成します。

```
$ ./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
```

- *AMDTClassicMatMul-bin* アプリケーションを起動します。IBS サンプルを収集し、生データ ファイルから IMIX レポートを生成します。

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

- *AMDTClassicMatMul-bin* を起動します。Assess Performance のプロファイリングを 10 秒間実行し、レポートを生成します。

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

- *AMDTClassicMatMul-bin* を起動します。SWP モードで IBS サンプルを収集し、ibs-op イベントで並べ替えたレポートを生成します。

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

- SWP モードで 10 秒間にわたって TBP サンプルを収集し、レポートを生成します。

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

- AMDTCClassicMatMul-bin* を起動します。コールスタック サンプリング付きで TBP を収集し、レポートを生成します。

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

- AMDTCClassicMatMul-bin* を起動して、コールスタック サンプリング付きで TBP を収集し (アンワインド FPO 最適化スタック)、レポートを生成します。

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

- AMDTCClassicMatMul-bin* を起動します。PMCx076 および PMCx0C0 のサンプルを収集し、レポートを生成します。

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

- AMDTCClassicMatMul-bin* を起動します。50000 の間隔で IBS オペレーションのサンプルを収集し、ibs-op イベントで並べ替えたレポートを生成します。

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

- スレッドに接続して、TBP サンプルを 10 秒間にわたって収集し、レポートを生成します。

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

- OpenMP アプリケーションの OpenMP トレース情報を収集し、-omp を渡し、レポートを生成します。

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

- MPI プロファイリング情報を収集し、レポートを生成します。

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

- PMCx076 および PMCx0C0 のサンプルを収集しますが、コールグラフ情報は PMCx0C0 のみに対して収集し、レポートを生成します。

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

- AMDTCClassicMatMul-bin* を起動します。すべての OS トレース イベントを収集し、レポートを生成します。

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

- AMDTCClassicMatMul-bin* を起動します。Host Identity Protocol (HIP) ドメインの GPU トレースを収集し、レポートを生成します。

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

- AMDTCClassicMatMul-bin* を起動します。TBP サンプル、HIP ドメインの GPU トレースを収集し、レポートを生成します。

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

- AMDTCClassicMatMul-bin* を起動します。GPU サンプル、OS トレースを収集し、レポートを生成します。

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

6.10 Info コマンド

このコマンドを使用すると、システムに関する一般情報、PMC イベントの詳細、事前定義イベントの詳細などを取得できます。

構造:

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

一般的な使用法:

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

6.10.1 オプション

次の表に、info コマンドのオプション一覧を示します。

表 39. AMDuProfCLI Info コマンドのオプション

オプション	説明
-h --help	ヘルプ情報を表示します。
--list <type>	次に示すそれぞれのタイプでサポートされる要素の一覧を表示します。 - collect-configs: collect --config オプションで利用できる事前定義プロファイル設定。 - predefined-events: collect --event オプションでの使用がサポートされる事前定義イベントのリスト。 - pmu-events: collect --event オプションで利用できる生の PMC イベント。または、info --pmu-event all を使用すると、サポートされるすべてのイベントの情報を出力できます。 - cacheline-events: キャッシュ解析のために report --sort-by オプションで利用できるイベント エイリアスのリスト。これは、Windows および Linux プラットフォームのみでサポートされています。 - view-configs: report --view オプションでの使用がサポートされるデータ ビュー設定のリスト。
--collect-config <name>	collect --config <name> オプションで使用する特定のプロファイル設定の詳細を表示します。 サポートされるプロファイル設定の詳細を表示するには、info --list collect-configs コマンドを使用します。
--view-config <name>	レポート生成オプション report --view <name> で使用される特定のビュー設定の詳細を表示します。 サポートされるデータ ビュー設定の詳細を表示するには、info --list view-configs コマンドを使用します。
--pmu-event <event>	指定された PMU イベントの詳細を表示します。サポートされる PMC イベントの一覧を表示するには、info --list pmu-events コマンドを使用します。
--system	このシステムのプロセッサ情報を表示します。

次の表に、Linux 専用の info コマンド オプションの一覧を示します。

表 40. AMDuProfCLI の Info コマンド – Linux 専用オプション

オプション	説明
--list <type>	次に示すそれぞれのタイプでサポートされる要素の一覧を表示します。 • trace-events: collect --trace os または collect --trace user オプションで使用できるトレース イベントのリスト。 • gpu-events: gpu プロファイル設定で使用できる GPU イベントのリスト。
--bpf	BPF サポートと BCC インストールの詳細を表示します。

6.10.2 例

それぞれの目的に応じて次のコマンドを使用します。

- システムの詳細を出力します。

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

- 事前定義プロファイルの一覧を出力します。

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

- PMU イベントの一覧を出力します。

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

- 事前定義レポート ビューの一覧を出力します。

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

- 「assess_ext」などの事前定義プロファイルの詳細を出力します。

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

- PMCx076 などの PMU イベントの詳細を出力します。

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

- ibs_op_overall などのビュー設定の詳細を出力します。

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

- トレース イベントの一覧を出力します。

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

第 7 章 パフォーマンス解析

7.1 CPU プロファイリング

AMD uProf の CPU プロファイラーは、サンプリングに基づく統計アプローチに従ってプロファイル データを収集し、アプリケーション内のパフォーマンス ボトルネックを特定します。ここでは、CPU プロファイラーの機能を理解するために大まかな機能をいくつか示します。

- プロファイル データの収集には、次のいずれかのアプローチが使用されます。
 - 時間ベース プロファイリング (TBP) - プロファイリング対象アプリケーションに含まれるホットスポットを特定します。
 - イベント ベース プロファイリング (EBP) - コア PMC イベントのサンプリングにより、プロファイリング対象アプリケーションに含まれるマイクロ アーキテクチャに関連するパフォーマンスの問題を特定します。
 - 命令ベース サンプリング (IBS) - 命令ベースの高精度サンプリング。
- コールスタック サンプリング
- セカンダリ プロファイル データ
 - スレッド同時実行性 (Windows のみ。管理者権限が必要)
 - スレッド名 (Windows と Linux のみ)
- プロファイル スコープ
 - アプリケーションの起動 - アプリケーションを起動して、そのプロセスおよび子プロセスをプロファイリングします。
 - システム全体 - 実行中のすべてのプロセスおよび/またはカーネルをプロファイリングします。
 - プロセスの接続 - 既存アプリケーションに接続します (ネイティブ アプリケーションのみ)。
- プロファイル モード
 - ユーザー/カーネル - アプリケーションがユーザー モードおよび/またはカーネル モードで実行されているとき、プロファイル データが収集されます。
- サポートされる言語
 - C、C++
 - Java
 - .NET (5.0、6.0、Framework)
 - FORTRAN
 - アセンブリ アプリケーション

- サポートされるソフトウェア コンポーネント
 - ユーザー スペース アプリケーション
 - 動的にリンク/ロードされるモジュール
 - ドライバー
 - OS カーネル モジュール
 - プロファイル データの原因はさまざまな粒度で特定されます。
 - プロセス、スレッド、ロード モジュール、関数、ソース行、またはディスアセンブリ
 - C++ と Java のインライン関数
- 注記:** uProf でプロファイル データを関数およびソース行に関連付けるには、コンパイラによって生成されるデバッグ情報が必要です。
- データおよびレポート ファイル
 - 収集されたプロファイル データは、初めに生データ ファイルに格納されます。
 - 処理済みのプロファイル データは、CLI レポートまたは GUI の視覚化を生成するために使用されるデータベース ファイルに格納されます。
 - プロファイル レポートはカンマ区切り値 (CSV) 形式のファイルに保存され、任意のスプレッドシート ビューアーを使用して表示できます。
 - コマンド ライン インターフェイスの AMDuProfCLI を使用すると、プロファイル実行の設定、プロファイル データの収集、プロファイル レポートの生成が可能です。
 - 収集コマンドにより、プロファイル データを設定および収集できます。
 - レポート コマンドにより、プロファイル データを処理して、プロファイル レポートを生成できます。
 - プロファイル コマンドにより、パフォーマンス プロファイル データを収集して解析し、プロファイル レポートを生成できます。
 - AMDuProf GUI は次の目的に使用できます。
 - プロファイル実行を設定する。
 - プロファイル実行を開始し、パフォーマンス データを収集する。
 - パフォーマンス データを解析し、潜在的なボトルネックを特定する。

- 次に示すように、AMDuProf GUI には各種の UI 要素が含まれており、さまざまな粒度でプロファイルデータを解析して表示できます。
 - ホットスポット サマリ
 - セッション情報
 - スレッド同時並行性グラフ (Windows のみ。管理者権限が必要)
 - プロセスと関数の解析
 - ソースとディスアセンブリの解析
 - トップダウンおよびボトムアップ コールパス - アプリケーションの関数呼び出しフローを調べて、関数とその呼び出し先で消費された時間を解析するための視覚化。
 - フレーム グラフ - フレーム グラフとしてのコールスタックの視覚化
 - コールグラフ - コールスタックおよび呼び出し元/呼び出し先の表形式での視覚化
 - HPC - OpenMP と MPI のプロファイル データの解析
 - タイムライン ビジュアライザー - MPI API トレースおよび OS トレース情報のタイムライン ビュー
 - キャッシュ解析 - 偽共有されたホットなキャッシュ ラインの解析
- プロファイル制御 API
 - インストルメント化により、ターゲット アプリケーションからプロファイリングを個別に有効または無効にして、プロファイリングの範囲を制限します。

7.2 時間ベース プロファイリングによる解析

この解析では、指定された OS タイマー間隔に基づいて、定期的にプロファイル データが収集されます。これを使用して、プロファイル対象アプリケーションで最も多くの時間を消費しているホットスポットを特定できます。このようなホットスポットは、より詳しい調査と最適化のための良い候補となります。

7.2.1 プロファイリングの設定と開始

プロファイリングを設定して開始するには、次の手順に従います。

1. **[PROFILE]** → **[Start Profiling]** をクリックして、**[Select Profile Target]** 画面に移動します。
2. 目的のプロファイル ターゲットを選択し、**[Next]** ボタンをクリックします。
[Select Profiling] 画面が表示されます。

3. **[Select Profiling]** 画面で、**[Predefined Configs]** タブを選択します。

次の画面が表示されます。

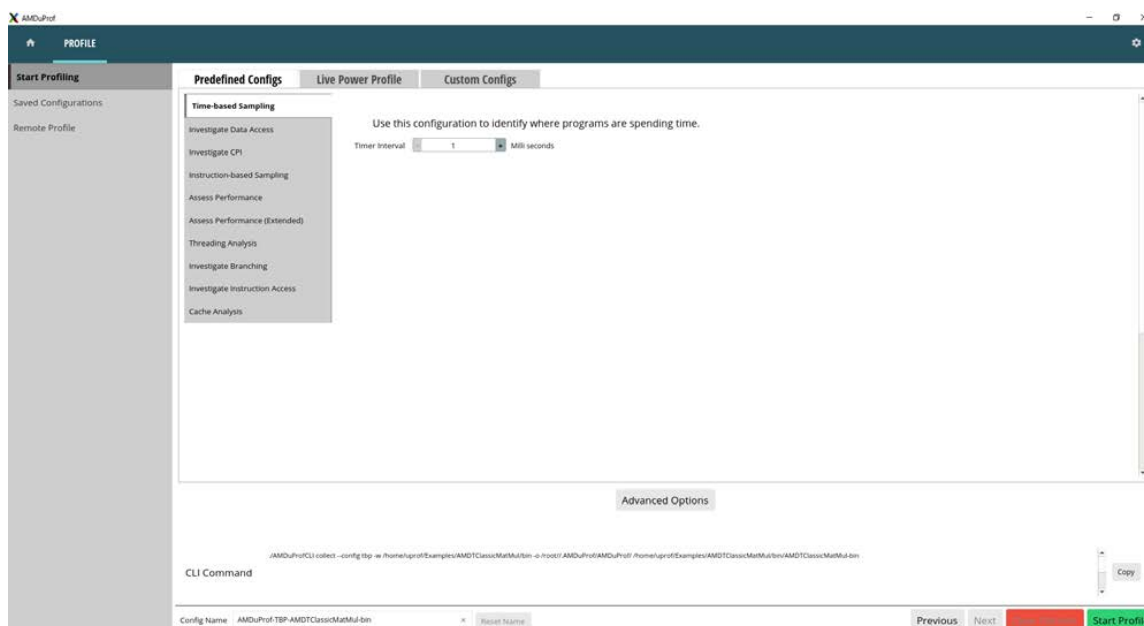


図 34. 時間ベース プロファイル - 設定

4. 左側の縦型ペインで **[Time-based Sampling]** を選択します。
5. **[Advanced Options]** をクリックして、コールスタックを有効にし、シンボルパス (デバッグ ファイルが別の場所にある場合) とその他のオプションを設定します。この画面の詳細は、[「Advanced Options」](#) を参照してください。
6. すべてのオプションを設定すると、一番下の **[Start Profile]** ボタンが有効になり、これをクリックするとプロファイリングを開始できます。

プロファイルの初期化の後で、プロファイル データの収集画面が表示されます。

7.2.2 プロファイル データの解析

プロファイル データを解析するには、次の手順に従います。

1. プロファイリングが停止すると、収集された生プロファイル データが自動的に処理され、**[Summary]** ページの **[Hot Spots]** 画面が表示されます。ホットスポットは、**Timer** サンプルに対して表示されます。この画面の詳細は、[「パフォーマンス ホットスポットの概要」](#) を参照してください。
2. 上部の水平ナビゲーション バーで **[ANALYZE]** をクリックして、**[Function HotSpots]** 画面に移動します。この画面の詳細は、[「関数ホットスポット」](#) を参照してください。
3. **[ANALYZE]** → **[Metrics]** をクリックすると、プロセス、ロード モジュール、スレッド、関数といった各種の粒度でプロファイル データ テーブルを表示できます。この画面の詳細は、[「プロセスと関数」](#) を参照してください。

4. **[Metrics]** 画面の **[Functions]** テーブルで任意のエントリをダブルクリックすると、**[SOURCES]** ページに関数のソース タブが読み込まれます。この画面の詳細は、「[ソースとアセンブリ](#)」を参照してください。

7.3 イベント ベース プロファイリングによる解析

このプロファイリングでは、AMD uProf は PMC を使用して、AMD x86 ベース プロセッサでサポートされる各種のマイクロ アーキテクチャ イベントを監視します。プロファイル対象アプリケーションに含まれる CPU およびメモリ関連のパフォーマンスの問題を特定するのに役立ちます。

7.3.1 プロファイリングの設定と開始

プロファイリングを設定して開始するには、次の手順に従います。

1. **[PROFILE]** → **[Start Profiling]** をクリックして、**[Select Profile Target]** 画面に移動します。
2. 目的のプロファイル ターゲットを選択し、**[Next]** ボタンをクリックします。

次のように、**[Start Profiling]** 画面が表示されます。

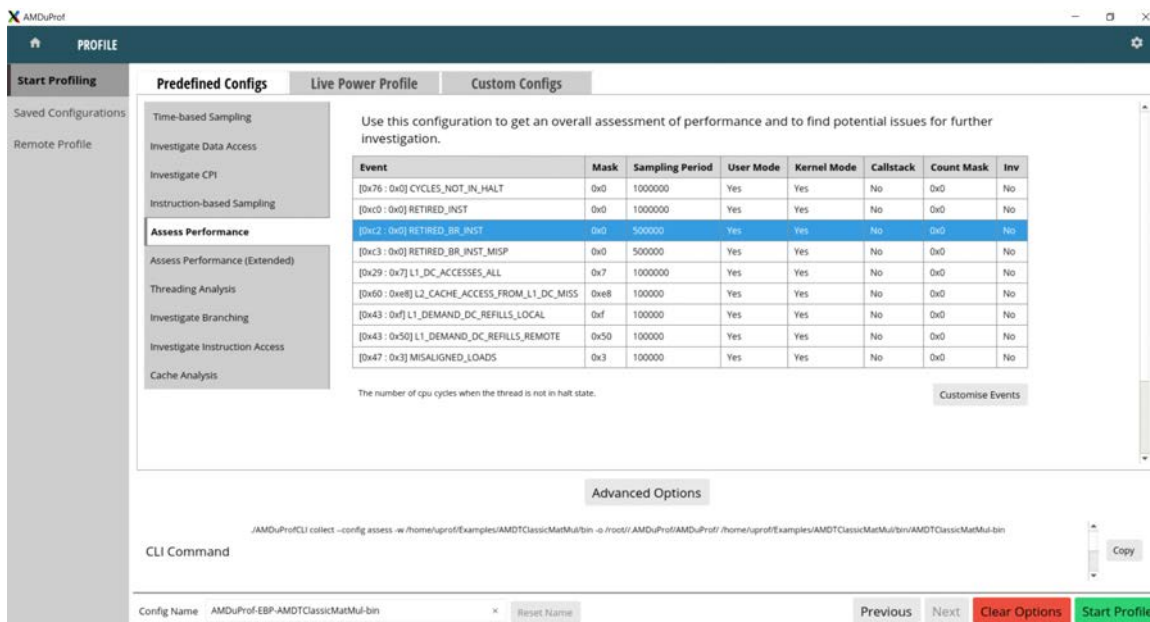


図 35. イベント ベース プロファイル - 設定

3. **[Select Profiling]** 画面で、**[Predefined Configs]** タブを選択します。
4. 左側の縦型ペインで **[Assess Performance]** を選択します。EBP に基づく事前定義サンプリング設定の詳細は、「[事前定義サンプリング設定](#)」を参照してください。
5. **[Advanced Options]** をクリックして、コールスタックを有効にし、シンボルパス (デバッグ ファイルが別の場所にある場合) とその他のオプションを設定します。この画面の詳細は、「[Advanced Options](#)」を参照してください。

- すべてのオプションを設定したら、一番下の **[Start Profile]** ボタンが有効になります。これをクリックしてプロファイリングを開始します。

プロファイルの初期化の後で、プロファイル データの収集画面が表示されます。

7.3.2 プロファイル データの解析

プロファイル データを解析するには、次の手順に従います。

- プロファイリングが停止すると、収集された生プロファイル データが自動的に処理され、**[Summary]** ページの **[Hot Spots]** 画面が表示されます。この画面の詳細は、「[パフォーマンス ホットスポットの概要](#)」を参照してください。
- 上部の水平ナビゲーション バーで **[ANALYZE]** をクリックして、**[Function HotSpots]** 画面に移動します。この画面の詳細は、「[関数ホットスポット](#)」を参照してください。
- [ANALYZE]** → **[Metrics]** をクリックすると、プロセス、ロード モジュール、スレッド、関数といった各種の粒度でプロファイル データ テーブルを表示できます。この画面の詳細は、「[プロセスと関数](#)」を参照してください。
- [Grouped Metrics]** 画面の **[Functions]** テーブルで任意のエントリをダブルクリックすると、**[SOURCES]** ページに関数のソース タブが読み込まれます。この画面の詳細は、「[ソースとアセンブリ](#)」を参照してください。

7.4 命令ベース サンプリングによる解析

このプロファイルで、AMD uProf は、AMD x64 ベース プロセッサでサポートされる IBS を使用して、ホットスポットに含まれるパフォーマンスの問題を診断します。また、プロセッサとメモリ サブシステムで命令がどのように動作しているかに関するデータを収集します。

7.4.1 プロファイリングの設定と開始

プロファイリングを設定して開始するには、次の手順に従います。

- [PROFILE]** → **[Start Profiling]** をクリックして、**[Select Profile Target]** 画面に移動します。
- 目的のプロファイル ターゲットを選択し、**[Next]** ボタンをクリックします。
- [Select Profiling]** 画面で、**[Predefined Configs]** タブを選択します。

4. 左側の縦型パネルで **[Instruction-based Sampling]** を選択します。IBS に基づく事前定義サンプリング設定の詳細は、「[事前定義サンプリング設定](#)」を参照してください。

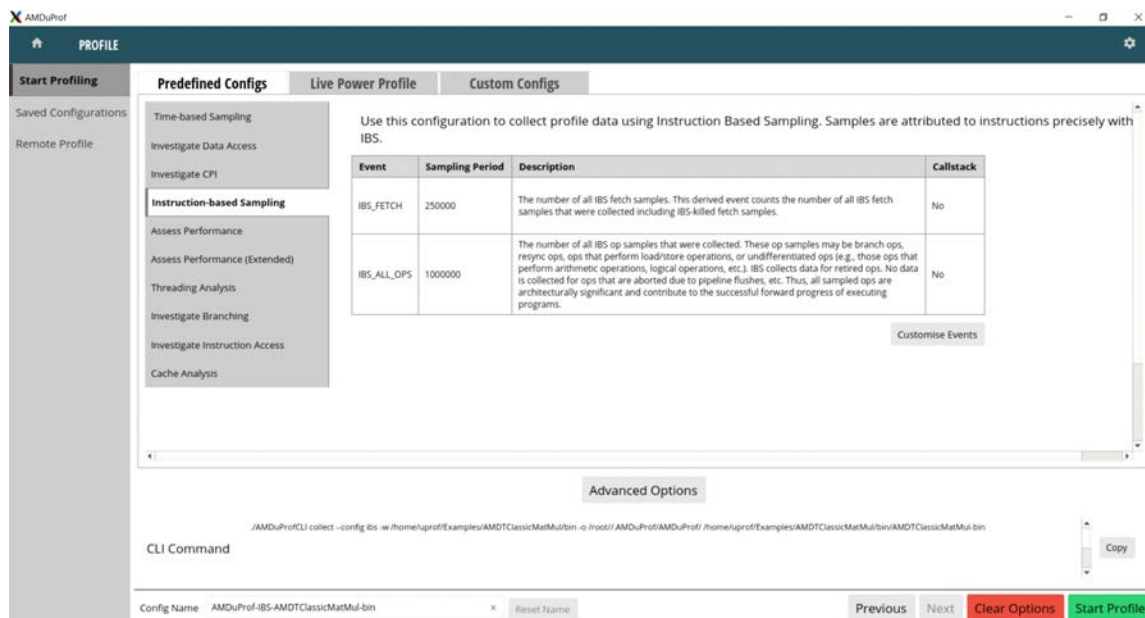


図 36. IBS の設定

5. **[Advanced Options]** をクリックして、コールスタックを有効にし、シンボルパス (デバッグ ファイルが別の場所にある場合) とその他のオプションを設定します。この画面の詳細は、「[Advanced Options](#)」を参照してください。
6. すべてのオプションを設定したら、一番下の **[Start Profile]** ボタンが有効になります。これをクリックしてプロファイリングを開始します。

プロファイルの初期化の後で、プロファイル データの収集画面が表示されます。

7.4.2 プロファイル データの解析

プロファイル データを解析するには、次の手順に従います。

1. プロファイリングが停止すると、収集された生プロファイル データが自動的に処理され、**[Summary]** ページの **[Hot Spots]** 画面が表示されます。この画面の詳細は、「[パフォーマンス ホットスポットの概要](#)」を参照してください。
2. 上部の水平ナビゲーション バーで **[ANALYZE]** をクリックして、**[Function HotSpots]** 画面に移動します。この画面の詳細は、「[プロセスと関数](#)」を参照してください。
3. **[ANALYZE]** → **[Metrics]** をクリックすると、プロセス、ロード モジュール、スレッド、関数といった各種の粒度でプロファイル データ テーブルを表示できます。この画面の詳細は、「[プロセスと関数](#)」を参照してください。
4. **[Grouped Metrics]** 画面の **[Functions]** テーブルで任意のエントリをダブルクリックすると、**[SOURCES]** ページに関数のソース タブが読み込まれます。この画面の詳細は、「[ソースとアセンブリ](#)」を参照してください。

7.5 コールスタック サンプルによる解析

コールスタック サンプルは、すべての CPU プロファイル タイプで、C、C++、Java アプリケーションに対して収集できます。これらのサンプルは、フレーム グラフ ウィンドウとコールグラフ ウィンドウの表示に使用されます。

注記: Java コールスタック プロファイリングがサポートされるのは、Linux プロファイルのみです。

コールスタック サンプリングを有効にするには、次の手順に従います。

1. プロファイル ターゲットとプロファイル タイプを選択します。
2. **[Advanced Options]** ボタンをクリックし、**[Call Stack Options]** ペインの **[Enable CSS]** オプションをオンにします。

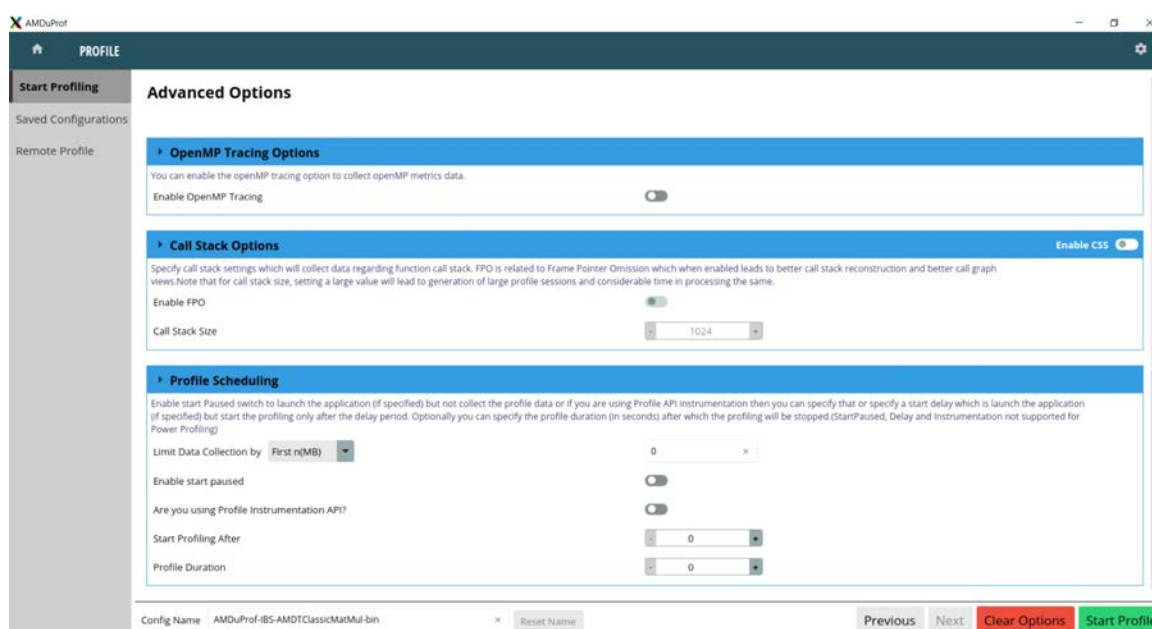


図 37. [Start Profiling] - [Advanced Options]

この画面の詳細は、「[Advanced Options](#)」を参照してください。

注記: アプリケーションが高い最適化レベルでコンパイルされており、フレーム ポインターが表示されない場合は、**[Enable FPO]** オプションをオンにできます。Linux の場合、これにより生プロファイル ファイルのサイズが大きくなります。

7.5.1 フレーム グラフ

フレーム グラフは、コールスタック サンプルに基づいてスタックを視覚化したグラフです。[Flame Graph] は [ANALYZE] ページに表示され、コールスタック サンプルを解析してホットなコールパスを特定するために使用できます。フレーム グラフにアクセスするには、[ANALYZE] から左側の縦型ペインの [Flame Graph] に移動します。

この画面の詳細は、「[フレーム グラフ](#)」を参照してください。

次の図に、フレーム グラフの例を示します。

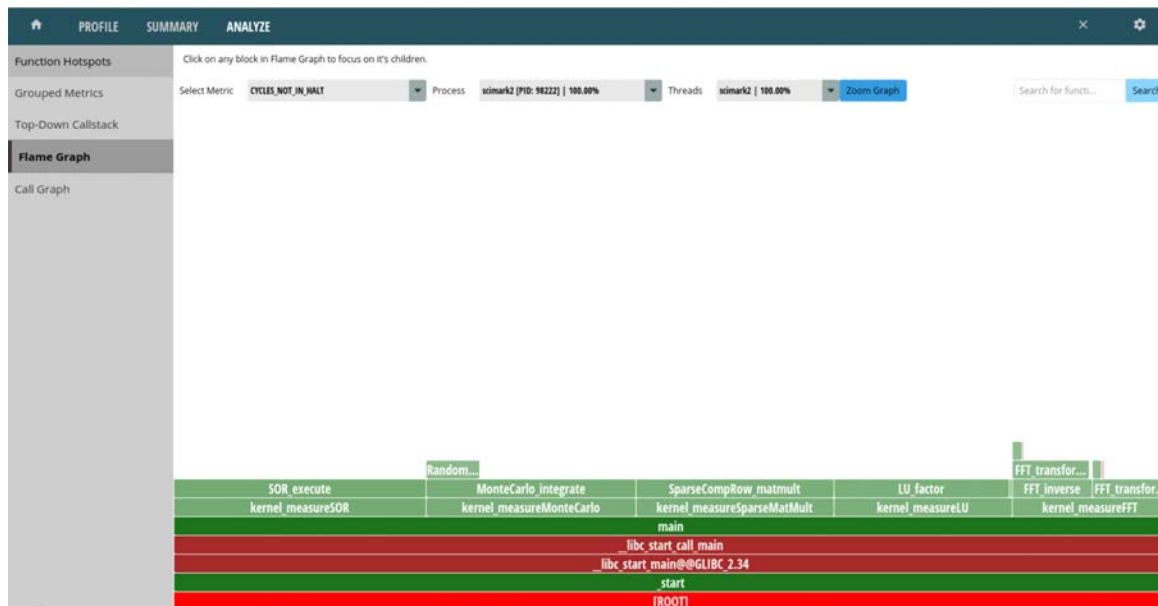


図 38. [ANALYZE] - [Flame Graph]

フレーム グラフは、[Process] ドロップダウンと [Select Metric] ドロップダウンに基づいて表示できます。また、関数検索ボックスで特定の関数名を検索すると、ハイライト表示できます。

7.5.2 コールグラフ

コールグラフには、コールスタック サンプルに基づくコールグラフのバタフライ ビューが表示されます。**[CallGraph]** は **[ANALYZE]** ページに表示され、コールスタック サンプルを解析してホットなコールパスを特定するために使用できます。コールグラフにアクセスするには、**[ANALYZE]** から左側の縦型パネルの **[Call Graph]** に移動します。

この画面の詳細は、「[コールグラフ](#)」を参照してください。

次の図に、コールグラフの例を示します。

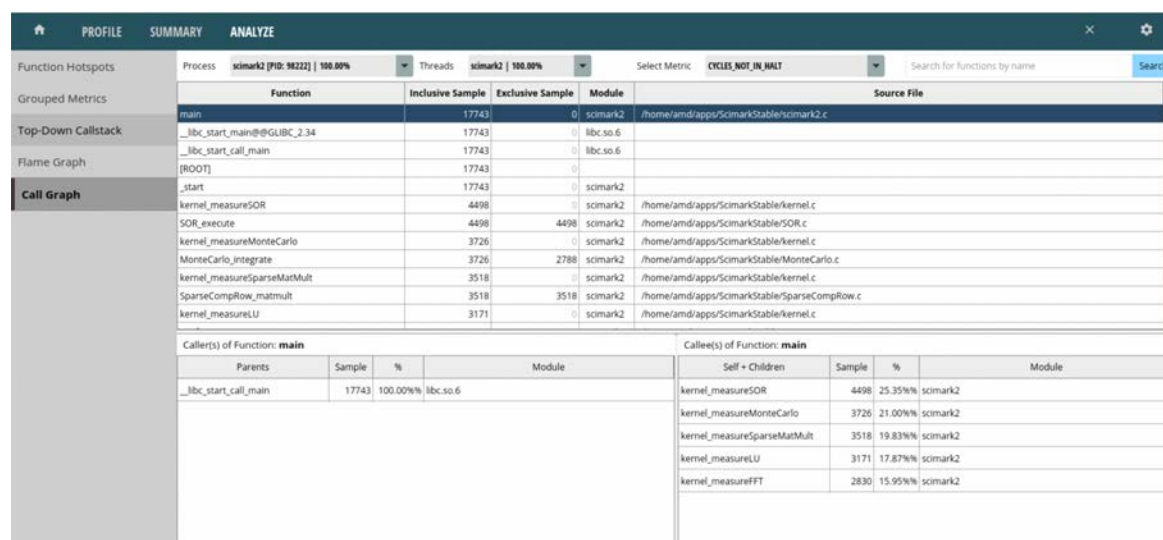


図 39. [ANALYZE] - [Call Graph]

[Process] ドロップダウンと **[Select Metric]** ドロップダウンに基づくデータを表示できます。上部中央のテーブルには、関数ごとのコールスタック サンプルが表示されます。いずれかの関数をクリックすると、下部にある 2 つのテーブル **[Caller(s)]** と **[Callee(s)]** が更新されます。これらのテーブルには、それぞれ、選択された関数の呼び出し元と呼び出し先が表示されます。

7.6 Java アプリケーションのプロファイリング

AMD uProf は、JVM 上で動作する Java アプリケーションのプロファイリングをサポートしています。このために、JVM Tool Interface (JVMTI) を使用します。

AMD uProf は、JVMTI エージェント ライブラリとして、Windows では *AMDJvmtiAgent.dll* を、Linux では *libAMDJvmtiAgent.so* を提供しています。この JvmtiAgent ライブラリは、ターゲットの JVM プロセスの起動中に読み込まれる必要があります。

7.6.1 Java アプリケーションの起動

Java アプリケーションが AMD uProf によって起動された場合、Java `-agentpath` オプションを使用して、ツールから JVM に AMDJvmtiAgent ライブラリが渡されます。AMD uProf は、プロファイル データを収集して、サンプルの原因となるインタープリット済みの Java 関数を特定できます。

Java アプリケーションをプロファイリングするには、次のサンプル コマンドを使用します。

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

レポートを生成するには、次のソース ファイル パスを指定します。

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

7.6.2 Java プロセスからプロファイルへの接続

AMD uProf では、既に実行中の JVM には JvmtiAgent を動的に接続できません。このため、接続プロセス メカニズムによってプロファイリングされる JVM プロセスの場合、JVM プロセスの起動中に JvmtiAgent ライブラリが読み込まれない限り、AMD uProf からクラス情報を取り込むことはできません。

実行中の Java プロセスをプロファイリングするには、Java アプリケーションの起動中に `-agentpath <path to agent lib>` オプションを渡します。こうすることで、AMD uProf を Java PID に接続できるため、後からプロファイル データを収集できます。

Linux 上の 64 ビット JVM の場合

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

Windows 上の 64 ビット JVM の場合

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

上に示した JVM インスタンスのプロセス ID (PID) をメモします。その後 AMD uProf GUI または AMD uProf CLI を起動し、このプロセスとプロファイルに接続します。

7.6.3 Java ソース ビュー

AMD uProf によって、プロファイル サンプルの原因となる Java メソッドが特定されると、ソース タブが開き、Java のソース行と、その行に起因する該当サンプルが表示されます。

ソース画面の詳細は、「[ソースとアセンブリ](#)」を参照してください。

次の図に、Java メソッドのソース ビューを表示します。

The screenshot shows the AMD uProf interface with the 'SOURCES' tab selected. It displays the source code for the method `jnt.scimark2.SparseCompRow::matmult(double[], double[], int[], int[], double[], int)`. The source code is shown in a table with columns for Line, Source, RETIRED_INST, CYCLES_NOT_IN_HALT, HE_ACCESS_FROM_L1, MISALIGNED_LOAD, RETIRED_BR_INST, and IPC. Below the source code, the assembly code is shown in a table with columns for Address, Line, Assembly, RETIRED_INST, CYCLES_NOT_IN_HALT, HE_ACCESS_FROM_L1, MISALIGNED_LOAD, RETIRED_BR_INST, and IPC.

Line	Source	RETIRED_INST	CYCLES_NOT_IN_HALT	HE_ACCESS_FROM_L1	MISALIGNED_LOAD	RETIRED_BR_INST	IPC
40	<code>sum += x[col[i]] * val[i];</code>						
41	<code>y[r] = sum;</code>	251	168	47			1.49
38	<code>int rowRp1 = row[r+1];</code>	5526	2014	960			2.74
37	<code>int rowR = row[r];</code>	319	257	64			1.24
38	<code>int rowRp1 = row[r+1];</code>						
39	<code>for (int i=rowR; i<rowRp1; i++)</code>	6857	2472	1181			4.27
40	<code>sum += x[col[i]] * val[i];</code>	58421	20649	10041			27.83
41	<code>y[r] = sum;</code>						
42	<code>}</code>						
43	<code>}</code>						
44	<code>}</code>						

Address	Line	Assembly	RETIRED_INST	CYCLES_NOT_IN_HALT	HE_ACCESS_FROM_L1	MISALIGNED_LOAD	RETIRED_BR_INST	IPC
0x13f	40	<code>vaddsd xmm0,xmm0,xmm1</code>	5606	2117	972			2.65
0x143	40	<code>inc r10d</code>	7073	2517	1239			2.81
0x146	39	<code>cmp r10d,ebp</code>	14	10				1.40
0x149	39	<code>j1 0000000000000124h</code>	17					2.13
0x14b	39	<code>mov ebp,ecx</code>						
0x14d	39	<code>add ebp,fdh</code>	83	30	12			2.77
0x150	39	<code>cmp ecx,ebp</code>	15	13				1.15
0x152	39	<code>cmovl ebp,r14d</code>	928	284	165			3.27
0x156	39	<code>cmp r10d,ebp</code>	28	37				0.76
0x159	39	<code>jnl 00000000000001e5h</code>	2714	1042	496			2.60
0x15f	39	<code>nop</code>						
0x160	40	<code>mov eax,[rdi+r10*4+10h]</code>	43	38	4			1.13
0x165	40	<code>cmp eax,r9d</code>	48	52	9			0.92

図 40. Java メソッド - ソース ビュー

7.6.4 Java コールスタックおよびフレーム グラフ

注記: Java コールスタック プロファイリングがサポートされるのは、Linux プロファイルのみです。

Java アプリケーションのプロファイリングのためにコールスタックを収集するには、次のコマンドを使用します。

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

次の図に、Java アプリケーションのフレーム グラフを表示します。

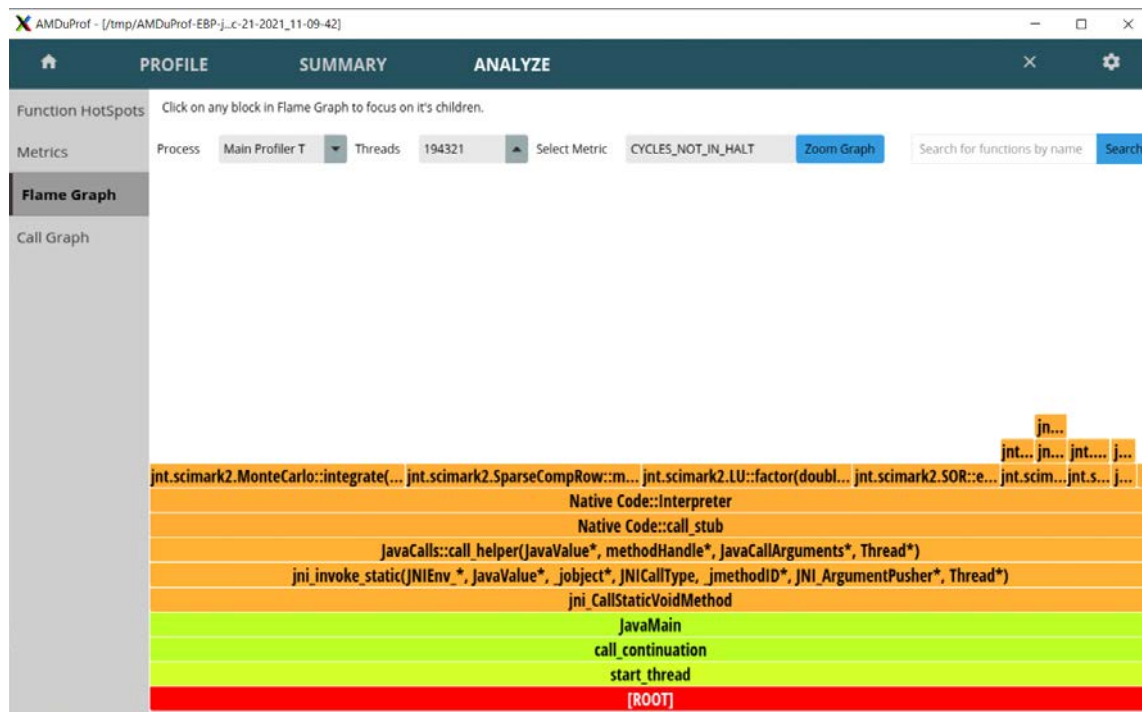


図 41. Java アプリケーション - フレーム グラフ

7.7 キャッシュ解析

キャッシュ解析では、IBS オペレーション サンプルを使用して、共有メモリ アプリケーションで、マルチスレッドおよびマルチプロセスに含まれるホットなキャッシュ ライン偽共有を検出します。

この機能は主に次の内容をレポートします。

- 偽共有の可能性のあるキャッシュ ライン
- アクセスが発生しているオフセット、そのオフセットへの読み出し関数と書き込み関数
- これらの読み出しおよび書き込み関数の PID、TID、関数名、ソース ファイル、行番号
- 該当キャッシュ ラインへのロードでのロード レイテンシ

7.7.1 サポートされるメトリクス

次に示す IBS オペレーション由来のメトリクスを使用して、キャッシュ偽共有レポートが生成されます。

表 41. IBS オペレーション由来のメトリクス

メトリクス	説明
IBS_LOAD_STORE	サンプリングされた合計ロードおよびストア
IBS_LOAD	合計ロード
IBS_STORE	合計ストア
IBS_DC_MISS_LAT	キャッシュ ラインへのロードでの蓄積ロード レイテンシ
IBS_LOAD_DC_L2_HIT	データ キャッシュまたは L2 キャッシュでのロード オペレーション ヒット
IBS_NB_LOCAL_CACHE_MODIFIED	ローカル キャッシュ (L3) からサービスされ、キャッシュ ヒット ステートが Modified であったロード
IBS_NB_LOCAL_CACHE_OWNED	ローカル キャッシュ (L3) からサービスされ、キャッシュ ヒット ステートが Owned であったロード
IBS_NB_LOCAL_CACHE_MISS	ローカル キャッシュ (L3) でミスし、リモート キャッシュ、ローカル またはリモート DRAM からサービスされたロード
IBS_NB_REMOTE_CACHE_MODIFIED	リモート キャッシュ (L3) からサービスされ、キャッシュ ヒット ステートが Modified であったロード
IBS_NB_REMOTE_CACHE_OWNED	リモート キャッシュ (L3) からサービスされ、キャッシュ ヒット ステートが Owned であったロード
IBS_NB_LOCAL_DRAM	ローカル メモリ (ローカル ソケットまたはローカル CCD に接続されたメモリ チャネル) でヒットしたロード
IBS_NB_REMOTE_DRAM	リモート メモリ (リモート ソケットまたはローカル ソケット内のその他の CCD に接続されたメモリ チャネル) でヒットしたロード
IBS_STORE_DC_MISS	データ キャッシュでミスしたストア オペレーション

7.7.2 GUI を使用したキャッシュ解析

プロファイリングの設定と開始

キャッシュ解析を実行するには、次の手順に従います。

1. プロファイル ターゲットを選択します。
2. **[Predefined Configs]** タブで **[Cache Analysis]** プロファイル タイプを選択します。
3. プロファイリングを開始します。

レポートの解析

プロファイリングが完了したら、[MEMORY] タブの [Cache Analysis] ページに移動して、プロファイル データを解析します。このページには、キャッシュラインとそのオフセットが関連付けられたメトリクス値と共に表示されます。

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

図 42. [Cache Analysis]

[Cache Analysis] 画面には次のオプションがあります。

- [Group By] ドロップダウンにより、詳細テーブルでキャッシュライン サンプルがグループ化される方法が決まります。選択肢には、[Cache Line Offset] があります。
- [ValueType] ドロップダウンを使用して、サンプル数の値を表示できます。

7.7.3 CLI を使用したキャッシュ解析

CLI には、解析データをキャッシュするための設定タイプ「memory」があります。プロファイル データを収集するには、次のコマンドを実行します。

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

このコマンドによりプログラムが起動され、キャッシュ解析レポートの生成に必要なプロファイル データが収集されます。生プロファイル データ ファイルは、/tmp/cache_analysis/AMDuProf-IBS_<timestamp>/ ディレクトリ内に作成されます。

レポートの生成と解析

キャッシュ解析レポートを生成するには、次の CLI コマンドを使用します。

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

これにより、CSV レポートが /tmp/cache_analysis/AMDuProf-IBS_<timestamp>/report.csv として生成され、ファイル内に次のセクションが含まれます。

- **SHARED DATA CACHELINE SUMMARY:** すべてのメトリクスのサマリ値を示したリスト。
- **SHARED DATA CACHELINE REPORT:** キャッシュラインと関連するメトリクスのサマリ値を示したリスト。

- **SHARED DATA CACHELINE DETAIL REPORT:** 次を示したリスト。
 - 偽共有の可能性のあるキャッシュ ライン
 - アクセスが発生しているオフセット、そのオフセットへの読み出し関数と書き込み関数
 - これらの読み出しおよび書き込み関数の PID、TID、関数名、ソース ファイル、行番号
 - 該当キャッシュ ラインへのロードでのロード レイテンシ
 - サポートされるメトリクス

次の図に、**キャッシュ解析**のサマリ セクションを示します。

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_LOCAL_DRAM	IBS_NB_REMOTE_DRAM	IBS_NB_CACHE_MODIFIED	IBS_NB_LOCAL_CACHE_MODIFIED	IBS_NB_REMOTE_CACHE_MODIFIED	IBS_NB_LOCAL_CACHE_OWNED	IBS_NB_REMOTE_CACHE_OWNED	IBS_NB_LOCAL_CACHE_MISS		
0x3ecd4220c0	43639	6882	36757	38615	6540	36655	0	0	342	342	0	0	0	0	0	0
0xb617fa40c0	1902	182	1720	1393	170	1716	0	0	12	12	0	0	0	0	0	0
0xb617fa4000	7	7	0	0	7	0	0	0	0	0	0	0	0	0	0	0

図 43. キャッシュ解析 - サマリ セクション

次の図に、**キャッシュ解析**の詳細レポートを示します。

SHARED DATA CACHELINE DETAILED REPORT																
CacheLine Address	Offset	Thread Id	IBS_LOAD	IBS_LOAD	IBS_STORE	IBS_DC_MISS_LAT	IBS_LOAD_DC_L2_HIT	IBS_STORE_DC_MISS	IBS_NB_LOCAL_DRAM	IBS_NB_REMOTE_DRAM	IBS_NB_CACHE_MODIFIED	IBS_NB_LOCAL_CACHE_MODIFIED	IBS_NB_REMOTE_CACHE_MODIFIED	IBS_NB_LOCAL_CACHE_OWNED	IBS_NB_REMOTE_CACHE_OWNED	IBS_NB_LOCAL_CACHE_MISS
0x3ecd4220c0	0x8	55896	6882	6882	0	38615	6540	0	0	0	342	342	0	0	0	0
	0x10	55897	36757	0	36757	0	0	36655	0	0	0	0	0	0	0	0
0xb617fa40c0	0x8	55896	182	182	0	1393	170	0	0	0	12	12	0	0	0	0
	0x10	55897	1720	0	1720	0	0	1716	0	0	0	0	0	0	0	0
0xb617fa4000	0x18	55896	4	4	0	0	4	0	0	0	0	0	0	0	0	0
	0x18	55897	3	3	0	0	3	0	0	0	0	0	0	0	0	0

図 44. キャッシュ解析 - 詳細レポート

レポート生成中に並べ替え順を変更するには、次のいずれかのメトリクスを `--sort-by event=<METRIC>` (例: `--sort-by event=ldst-count`) オプションと共に使用します。

表 42. 並べ替え基準メトリック

並べ替え基準メトリック	説明
ldst-count	サンプリングされた合計ロードおよびストア
ld-count	合計ロード
st-count	合計ストア
cache-hitm	ローカルまたはリモート キャッシュ (L3) からサービスされ、キャッシュ ヒット ステートが Modified であったロード。

表 42. 並べ替え基準メトリック (続き)

並べ替え基準メトリック	説明
lcl-cache-hitm	ローカル キャッシュ (L3) からサービスされ、キャッシュ ヒット ステートが Modified であったロード
rmt-cache-hitm	リモート キャッシュ (L3) からサービスされ、キャッシュ ヒット ステートが Modified であったロード。
lcl-dram-hit	ローカル メモリ (ローカル ソケットまたはローカル CCD に接続されたメモリ チャンネル) でヒットしたロード
rmt-dram-hit	リモート メモリ (リモート ソケットまたはローカル ソケット内のその他の CCD に接続されたメモリ チャンネル) でヒットしたロード
l3-miss	ローカル キャッシュ (L3) でミスし、リモート キャッシュ、ローカルまたはリモート DRAM からサービスされるロード。
st-dc-miss	データ キャッシュでミスしたストア オペレーション

注記: `info --list cacheline-events` コマンドを使用すると、`sort-by` オプションでサポートされるメトリック一覧を表示できます。

7.8 カスタム プロファイル

事前定義設定のほかに、プロファイリングに必要なイベントを選択できます。カスタム プロファイリングを実行するには、次の手順に従います。

7.8.1 プロファイリングの設定と開始

1. **[PROFILE]** → **[Start Profiling]** をクリックして、**[Select Profile Target]** 画面に移動します。
2. 目的のプロファイル ターゲットを選択し、**[Next]** ボタンをクリックします。
[Select Profile Configuration] 画面が表示されます。
3. **[Select Profile Type]** ドロップダウンで、次のいずれかを選択します。
 - **[CPU Tracing Mode]** ドロップダウンには、**[OS Trace]** と **[User Mode Trace]** のオプションがあります。(サポートされるイベントに対して) **[OS Trace]** が有効になるのは、Linux ではルート/ADMIN モードが使用されている場合のみであり、Windows では、サポートされるイベント **[Schedule]** に対して有効になります。**[User Mode Trace]** は、Linux でのアプリケーション解析のみで有効です。

[CPU Trace] は次のように表示されます。

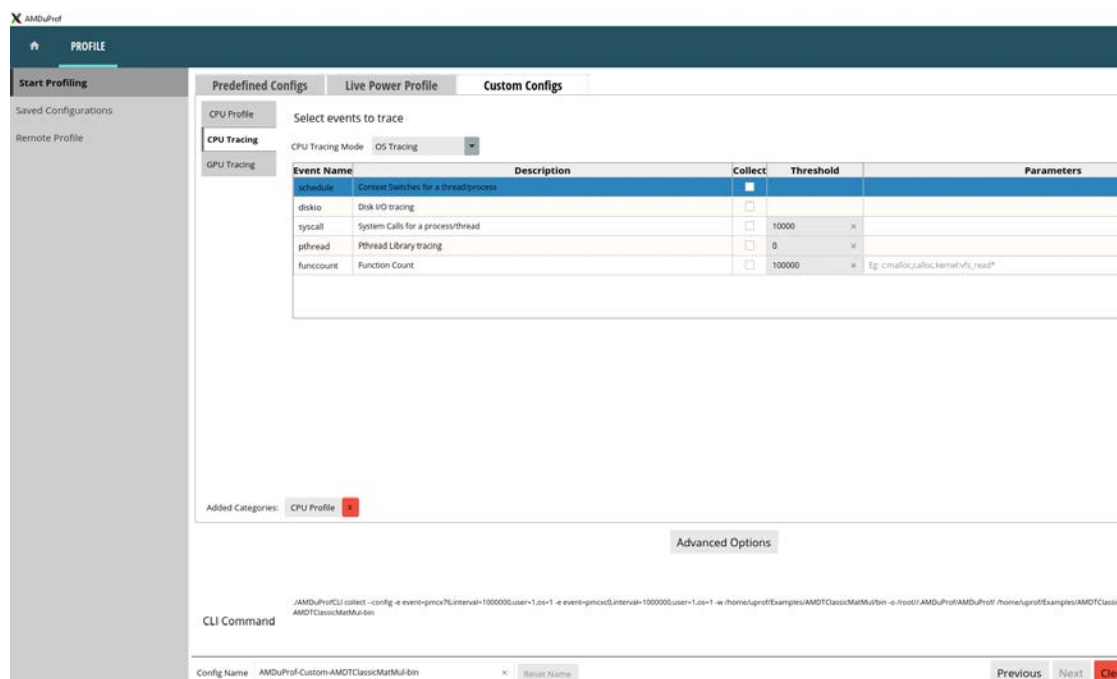


図 45. CPU トレース

– [GPU Trace] は次のように表示されます。

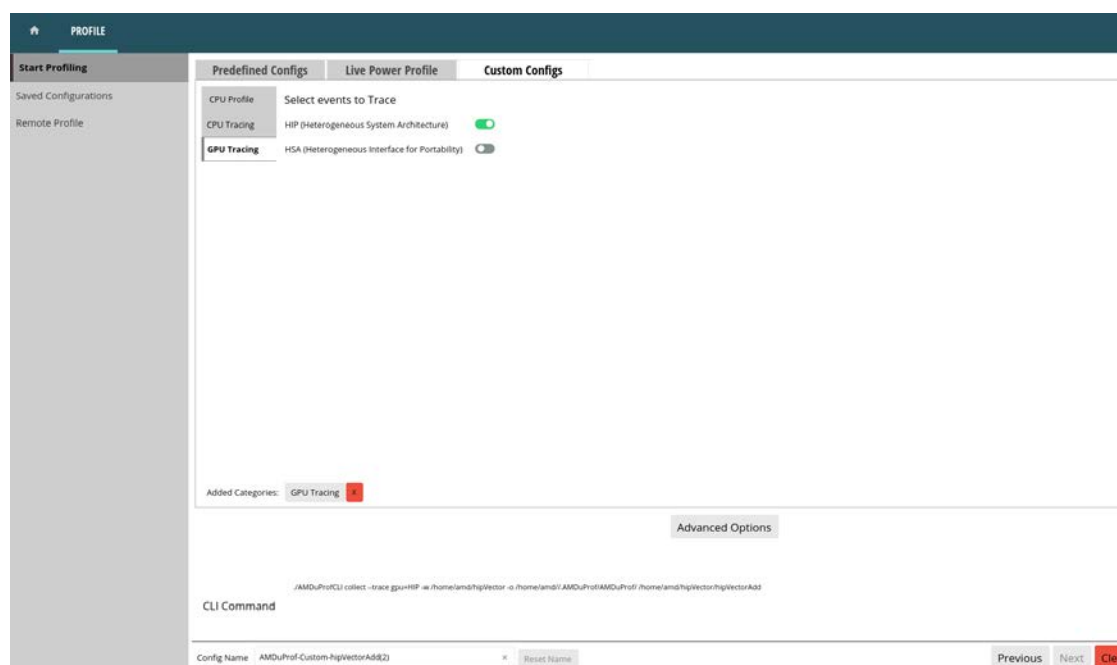


図 46. GPU トレース

CPU プロファイル + CPU トレースなど、カスタム設定から複数のカテゴリを同時に追加できます。

複数のカテゴリを選択すると、[Added Categories] にブレッドクラムとして表示され、不要なカテゴリの選択を解除できます。対応する CLI コマンドがその下に生成されます。

カスタム設定画面は、次のように表示されます。

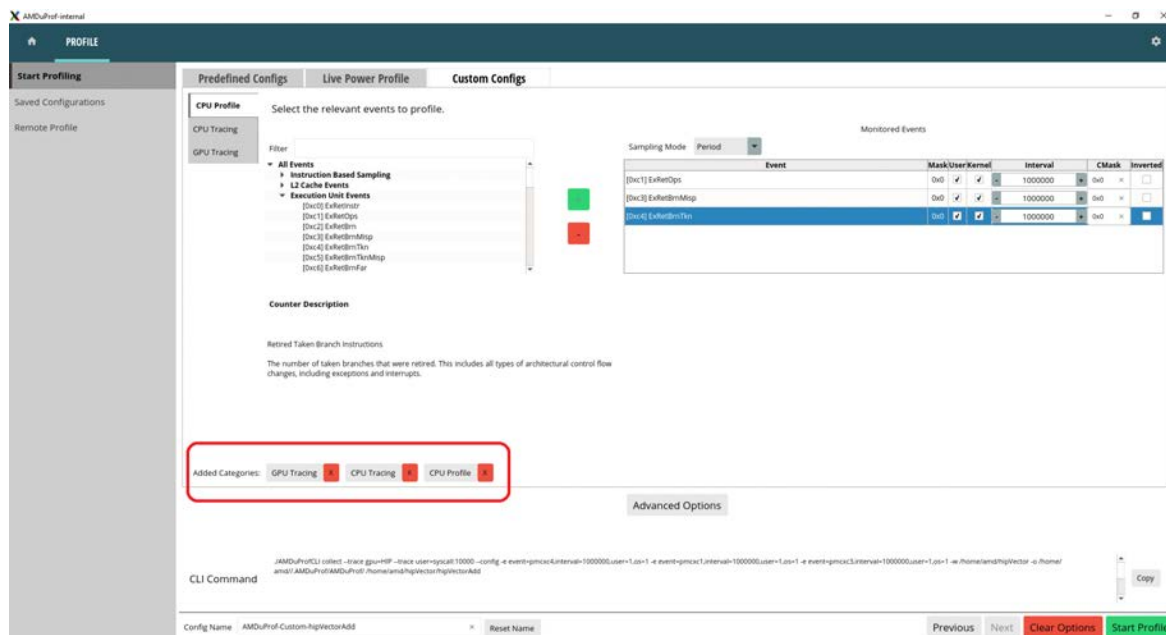


図 47. [Custom Config] - [Added Categories]

1. [Custom Configs] タブを選択し、左側の縦型ペインの [CPU Profile] を選択します。
2. [Advanced Options] をクリックして、コールスタックを有効にし、シンボルパス (デバッグファイルが別の場所にある場合) とその他のオプションを設定します。この画面の詳細は、「[Advanced Options](#)」を参照してください。
3. すべてのオプションを設定したら、一番下の [Start Profile] ボタンが有効になります。これをクリックしてプロファイリングを開始します。

プロファイルの初期化の後で、プロファイルデータの収集画面が表示されます。

7.8.2 プロファイルデータの解析

プロファイルデータを解析するには、次の手順に従います。

1. プロファイリングが停止すると、収集された生プロファイルデータが自動的に処理され、[Summary] ページの [Hot Spots] 画面が表示されます。この画面の詳細は、「[パフォーマンスホットスポットの概要](#)」を参照してください。
2. 上部の水平ナビゲーションバーで [ANALYZE] をクリックして、[Function HotSpots] 画面に移動します。この画面の詳細は、「[関数ホットスポット](#)」を参照してください。

3. **[ANALYZE]** → **[Metrics]** をクリックすると、プロセス、ロード モジュール、スレッド、関数といった各種の粒度でプロファイル データ テーブルを表示できます。この画面の詳細は、「[プロセスと関数](#)」を参照してください。
4. **[Metrics]** 画面の **[Functions]** テーブルで任意のエントリをダブルクリックすると、**[SOURCES]** ページに関数のソース タブが読み込まれます。この画面の詳細は、「[ソースとアセンブリ](#)」を参照してください。

7.9 注記

7.9.1 信頼性のしきい値

多重化または静的サンプリングが原因で、プログラム ユニットに対して収集されたサンプル数が少ないメトリクスは、グレーで表示されます。これに関して留意すべき点は次のとおりです。

- これは、SW タイマーとコア PMC に基づくメトリクスに当てはまります。
- この信頼性しきい値は、**[SETTINGS]** ページの **[Preferences]** セクションで設定できます。

7.9.2 問題のしきい値

特定のしきい値を超える (>1.0) CPI メトリクス セルは、ハイライトされます。このようなセルは次に示すようにピンク色で表示され、潜在的なパフォーマンスの問題として示されます。

PROFILE

SUMMARY

ANALYZE

図 48. CPI メトリクス - しきい値ベースのパフォーマンス

7.10 IBS サンプルの ASCII ダンプ

一部のシナリオでは、IBS オペレーション プロファイル サンプルの ASCII ダンプを解析することが役立ちます。実行するには、次の手順に従います。

1. IBS オペレーション サンプルを収集するには、次のコマンドを実行します。

```
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. 生ファイルが生成された後で次のコマンドを実行し、IBS オペレーション サンプルを変換して ASCII ダンプを取得します。

```
C:\> AMDuProfCLI.exe translate --ascii event-dump -i C:\temp\AMDuProf-IBS_<timestamp>\
```

IBS オペレーション サンプルの ASCII ダンプを含む CSV ファイルが生成されます。

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

3. 収集中、次の調整ノブを使用できます。

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

説明:

- interval はサンプリング間隔です
- loadstore は、ロードおよびストア オペレーションのみの収集を意味します (Windows のみのオプション)
- ibsop-count-control=1 は、ディスパッチされたマイクロ オペレーション数を表します (0 の場合は、クロック サイクル数)。
- --data-buffer-count 1024 は、コアあたりで割り当てるデータ バッファの数です (Windows のみのオプション)。

過度に多くのレコードが見つからない場合は、次の方法を試します。

- サンプリング間隔を大きくする
- データ バッファ数を増やす
- プロファイリング対象のコア数を減らす

7.11 分岐解析

AMD “Zen4” プロセッサは、Last Branch Record (LBR) と呼ばれる、分岐解析に役立つ CPU 機能をサポートしています。分岐解析レポートを収集して生成するには、uProf CLI を使用します。

分岐解析は、Linux プラットフォームのみでサポートされています。

注記:

1. LBR サンプルを収集するには、PMC イベントの有効化が必要です。PMC イベントが渡されない場合、LBR サンプル収集中に PMCX0C0 イベントが有効になります。
2. Java アプリケーションの分岐解析はサポートされていません。

例

LBR 情報の収集。

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

分岐解析レポートの生成。

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

サンプル レポート

生成されるレポートには、分岐解析用のセクションが含まれます。分岐解析サマリのサンプル スクリーンショットを次に示します。

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_64static	scimark2_64static	scimark2_64static	scimark2_64stat (PID:468487)	
18.02	1387483	0.01	77	SOR_execute	SOR_execute	/home/deesin/home/deesin/scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64stat (PID:468487)	
14.46	1113561	0	21	SparseCompRow_matmult	SparseCompRow_matmult	/home/deesin/home/deesin/scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64stat (PID:468487)	
5.35	411872	0.01	52	FFT_transform_internal	FFT_transform_internal	/home/deesin/home/deesin/scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64stat (PID:468487)	
4.4	338909	0	0	Random_nextDouble	Random_nextDouble	/home/deesin/home/deesin/scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64stat (PID:468487)	
3.54	272318	0	0	SparseCompRow_matmult	SparseCompRow_matmult	/home/deesin/home/deesin/scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64stat (PID:468487)	
2.45	188579	0.01	25	Random_nextDouble	MonteCarlo_integrate	/home/deesin/home/deesin/scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64stat (PID:468487)	
2.38	183204	0	8	FFT_transform_internal	FFT_transform_internal	/home/deesin/home/deesin/scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64stat (PID:468487)	
2.34	180103	0.01	25	Random_nextDouble	MonteCarlo_integrate	/home/deesin/home/deesin/scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64stat (PID:468487)	
2.33	179614	0	0	MonteCarlo_integrate	Random_nextDouble	/home/deesin/home/deesin/scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64stat (PID:468487)	
2.02	155651	0	0	MonteCarlo_integrate	Random_nextDouble	/home/deesin/home/deesin/scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64static	scimark2_64stat (PID:468487)	

図 49. 分岐解析サマリ

分岐解析サマリ テーブルには、次の列が含まれます。

- **OVERHEAD (%)**: 大部分が成立した分岐はどれかを示します。計算式は次のとおりです。 $(SAMPLES * 100) / (SAMPLES \text{ の合計})$ 。
- **SAMPLES**: その分岐のために収集されたサンプル数を表示します。実際に成立した分岐ではありません。
- **MISPREDICT (%)**: その分岐に対して発生した予測ミスの割合を表します。計算式は次のとおりです。 $((MISPREDICT \text{ COUNT}) * 100 / SAMPLES)$ 。
- **MISPREDICT COUNT**: その分岐のために収集されたサンプルのうち、分岐予測ミスしたサンプルの数を示します。
- **SOURCE FUNCTION**: 分岐が成立する前の関数を表示します。
- **TARGET FUNCTION**: 分岐が成立した後の関数を表示します。
- **SOURCE LINE**: SOURCE FUNCTION のファイルパスと (分岐が成立する前の) 行番号を表示します。
- **TARGET LINE**: TARGET FUNCTION のファイルパスと (分岐が成立した後の) 行番号を表示します。
- **SOURCE MODULE**: SOURCE FUNCTION のモジュール名を表示します。
- **TARGET MODULE**: TARGET FUNCTION のモジュール名を表示します。
- **PROCESS**: プロセスの名前と PID を表示します。

7.12 セッションのエクスポート

CLI オプション `--export-session` を使用すると、重要なセッション ファイルを含む圧縮アーカイブを生成できます。圧縮アーカイブは簡単にその他のシステムに転送でき、GUI を使用してパフォーマンス データを解析できます。

この機能により、複数システム間でのセッション ファイルの転送および利用プロセスが効率化されるため、より利用しやすくなり、ワークフローをスムーズに継続できるようになります。

手順

セッションをエクスポートするには、次の手順に従います。

1. `translate`、`report`、または `profile` コマンドを使用して、圧縮アーカイブを生成します。

`.zip` ファイルが 1 つ生成されます。

2. その `.zip` ファイルを別のシステムにコピーして解凍します。

解凍したセッション ディレクトリを GUI にインポートすると、データを視覚化して解析できます。解凍したセッションをインポートし、パフォーマンス データを解析する方法は、「[プロファイル データベースのインポート](#)」を参照してください。

一般的な使用法

- 「`translate`」コマンドを使用して圧縮アーカイブを生成します。

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

- 「`report`」コマンドを使用して圧縮アーカイブを生成します。

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

- 「`profile`」コマンドを使用して圧縮アーカイブを生成します。

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

例

エクスポート セッション オプションを有効にして、`AMDTClassicMatMul.exe` アプリケーションを起動し、時間ベース プロファイル (TBP) サンプルを収集し、レポートを生成します。

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

7.13 制限

AMD uProf での CPU プロファイリングには、次に示す制限事項があります。

- CPU プロファイリングを実行するには、プロファイル対象アプリケーションの実行ファイル バイナリが、VMProtect などの何らかのソフトウェア保護ツールによって、圧縮も難読化もされていないことが条件となります。
- 第 1 世代の AMD EPYC™ の B1 デバイスの場合、コア PMC のイベント ベース プロファイリング (EBP) で使用されるのは、一度に 1 つの PMC レジスタのみです。

IMIX には、次の制限事項があります。

- IMIX ビューまたはレポートがサポートされるのは、IBS プロファイル タイプに対してのみです。
- サンプル数が 10 未満のモジュール/バイナリがある場合、それらは IMIX レポートには表示されません。サンプル数が過度に少ない場合、IMIX 解析は役に立ちません。
- Linux カーネル モジュールの .ko ファイルは、IMIX ビューまたはレポートには表示されません。

第 8 章 パフォーマンス解析 (Linux)

この章では、Linux 固有のパフォーマンス解析モデルについて説明します。

8.1 スレッド解析

スレッド解析を使用して、次の要素に関してアプリケーション使用がどの程度効率的であるかを特定できます。

- プロセッサ コア
- 同期に起因するスレッド間の競合
- スレッドの CPU 使用率
- アプリケーション スレッドの実行時間と待機時間の解析

制限

- libc と libpthread が静的にアプリケーションにリンクされている場合、スレッド解析はサポートされません。
- スレッドまたはプロセスの作成で、アプリケーションが pthread_create() または fork() の代わりに、クローンシステムコールを使用する場合の動作は不定です。
- システム全体のプロファイリングでのプロセス接続はサポートされていません。
- AMD “Zen3” および AMD “Zen4” プラットフォームのみでサポートされます。その他のプラットフォームでは、カスタム設定を使用してデータを収集します。

8.1.1 CLI を使用したスレッド解析

AMDuProfCLI を使用して、必要なプロファイルおよびトレース データを収集し、詳しい分析のために .csv 形式でレポートを生成できます。また、処理済みのプロファイルおよびトレース データを GUI にインポートできます。

スレッド データの収集

スレッド データを収集するための CLI コマンドは次のとおりです。

```
$ 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
```

このコマンドを使用すると、プログラムが起動して、プロファイルおよびトレース データを収集します。起動アプリケーションが実行されると、AMDuProfCLI はセッションディレクトリパスを表示します。このディレクトリ内に、生のプロファイルおよびトレース データが保存されています。

上に示した例で、セッションディレクトリパスは次のとおりです。

/tmp/threading-analysis/AMDuProf-classic_lock-Threading_Jun-13-2023_06-00-23

スレッドとシステム コールの収集

各スレッドの I/O、システム コール、ブロック時間を取得するため、スレッドに加えてシステム コールの収集を有効にします。スレッドとシステム コールのトレースを収集するための CLI コマンドは、次のとおりです。

```
$ 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
```

スレッドとコンテキスト スイッチの収集

正確な待機時間解析を実行するには、コンテキスト スイッチの収集を有効にします (ルート アクセスが必要)。

```
$ 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
```

カスタム設定を使用した収集

周波数モード (周波数は 100 Hz に設定) での CPU サイクル イベント、pthread 同期 API トレース データ、システム コールを収集するためのサンプル コマンド。

```
$ 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
```

プロファイル レポートの生成

次の CLI report コマンドを使用し、セッション ディレクトリ パスをオプション -i の引数として渡して、.csv 形式でプロファイル レポートを生成します。

```
$ 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
```

データを処理してレポートを生成すると、ターミナルにレポート ファイル パスが表示されます。次に、.csv レポート ファイル内のトレース レポート セクションの例を示します。

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	

図 50. トレース レポート

レポートのアプリケーション パフォーマンス スナップショットでは、次の詳細が提供されます。

- スレッド数/スレッド カウント: アプリケーションによって作成されたスレッドの合計数。
- 経過時間: アプリケーションの合計経過時間。
- シリアル時間: 1 つのスレッドのみが実行されているアプリケーション時間の合計。
- 並列時間: 2 つ以上のスレッドが実行されているアプリケーション時間の合計。
- 実行時間: すべてのスレッドの合計実行時間。コンテキスト スイッチ レコードが収集されている場合、合計実行時間は、すべてのスレッドが CPU で実行されている時間の合計になります。それ以外の場合、合計実行時間 = 合計時間 - (合計待機時間 + 合計スリープ時間) です。

- 待機時間: すべてのスレッドの合計待機時間。待機時間の計算方法は次のとおりです。
 - スレッド設定 (`--config threading`): `pthread` 同期 API と待機システム コールに含まれるスレッドによって消費された合計時間。トレースされる同期 API と待機システム コールについては、セクション 8.1.2 および 8.1.3 を参照してください。
 - カスタム設定 (`--trace user=syscall`): 待機システム コールに含まれるスレッドによって消費された合計時間。
 - カスタム設定 (`--trace user=pthread`): `pthread` 同期 API に含まれるスレッドによって消費された合計時間。
 - カスタム設定 (`--trace os/--trace os=schedule`): スレッドが CPU 内にない場合のすべてのスレッドの合計時間。スレッドが CPU 内にあるかどうかを特定するために、コンテキスト スイッチ レコードが使用されます。
- スリープ時間: スリープ システム コールに含まれるすべてのスレッドによって消費された合計時間。トレースされるスリープ システム コールについては、8.1.3 を参照してください。
- I/O 時間: I/O システム コールに含まれるすべてのスレッドによって消費された合計時間。トレースされる I/O システム コールについては、8.1.3 を参照してください。
- ブロック時間: ブロック システム コールに含まれるすべてのスレッドによって消費された合計時間。アプリケーションがこのタイプのシステム コールを実行する場合、アプリケーションがブロックされる保証はありません。このため、このブロック時間は合計実行時間にも加えられます。トレースされるブロック システム コールについては、8.1.3 を参照してください。

サマリ レポートのセクション

- システム コール サマリ: システム コール カウント、アプリケーションによってシステム コールのために消費された合計時間を示します。多くの時間を消費しており、かつ、そのシステム コールが阻害要因となっている場合は最適化が見込まれるシステム コールを特定するのに役立ちます。
- スレッド サマリ: 合計実行時間、各スレッドの待機時間、スレッド合計時間に対する待機時間の割合を示します。スレッドが効果的にコアを使用しているかどうかを突き止めるのに役立ちます。アプリケーションを最適化するには、スレッドの待機時間を短くする必要があります。
- 待機オブジェクト サマリ: `pthread` 同期オブジェクトの待機カウントと、この同期オブジェクトに起因する合計待機時間。大半の待機時間の原因となるオブジェクトを特定するのに役立ちます。
- プロファイリングされたセッションを GUI にインポートし、[Analyze] → [Thread Timeline] に移動すると、データを視覚化して、スレッド タイムライン解析、`pthread` 同期オブジェクト解析、コールスタック解析を実行できます。

8.1.2 pthread 同期 API

pthread トレース イベントが有効な場合にトレースされるスレッド同期 API の一覧は、次のとおりです。

- 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 システム コール ラッパー API

システム コール イベントが有効な場合にトレースされる libc 関数の一覧は、次のとおりです。

スリープ API

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

待機 API

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

I/O API

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

ブロッキング API

- flock
- fsync
- sync
- syncfs
- fdatasync
- sync_file_range
- accept
- accept4
- recv
- recvfrom
- recvmsg
- recvmsg
- 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

その他の API

- 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
- fallocation
- ioperm
- iopl
- mount
- prctl
- ptrace
- sigaction
- swapon
- swapoff
- tee
- umount
- umount2
- unshare
- vhangup

8.1.4 Linux のタイムライン解析 GUI

GUI からスレッド解析を設定するには、次の手順に従います。

1. **[Select Profile Configuration]** 画面に移動します。
2. このタブで **[Predefined Configs]** を選択します。
3. 左側の縦型パネルの **[Threading Analysis]** を選択します。

CLI または GUI から収集したプロファイル データは、セッションをインポートすることにより、GUI で視覚化できます。インポートすると、次に示すセクション (Thread Timeline) が **[ANALYZE]** ページに表示されます。

時系列データは、エンティティ (スレッド、ランク、デバイスなど) ごとにタイムラインにプロットされます。タイムラインをズーム インした場合のみ、トレース データ (収集されている場合) がプロットされます。これは、データ サイズに関連するスケーラビリティの問題に対処するために使用できます (トレース データには非常に多数のレコードが含まれる可能性があり、すべてがプロットされている場合は視覚的に判読できません)。ビュー全体は、主に、上部のデータ セレクター、中央のタイムライン、下部のフィルターという 3 つの部分に縦に分かれています。タイムラインの使用法は、次のとおりです。

- カーソルをタイムラインに合わせると、特定のエンティティ向けのツール ヒントを含む縦線が表示され、関連する詳細と現在のタイムスタンプを確認できます。
- コールスタック データが収集されている場合、タイムライン内の任意の箇所をクリックすると、下部パネルに、対応するエンティティのコールスタックが表示されます。

注記: サンプリング データの粒度は粗いため、特定のタイムスタンプに複数のコールスタックが存在する場合があります。

- CPU プロファイル データが収集されている場合、タイムラインをクリックしてマウスをドラッグし、タイムライン全体から特定の領域を選択すると、選択した時間範囲内の **[Function Hotspot]** が表示されます。

- 水平方向にタイムラインをズームイン/アウトするには、次のいずれかの方法を使用します。
 - マウス ホイールを使用します。
 - キーボードの **Ctrl** と **+**/**-** キーを同時に押して、それぞれズームイン/アウトします。
- タイムラインにズームインすると、トレース データが表示されます (データがある場合)。

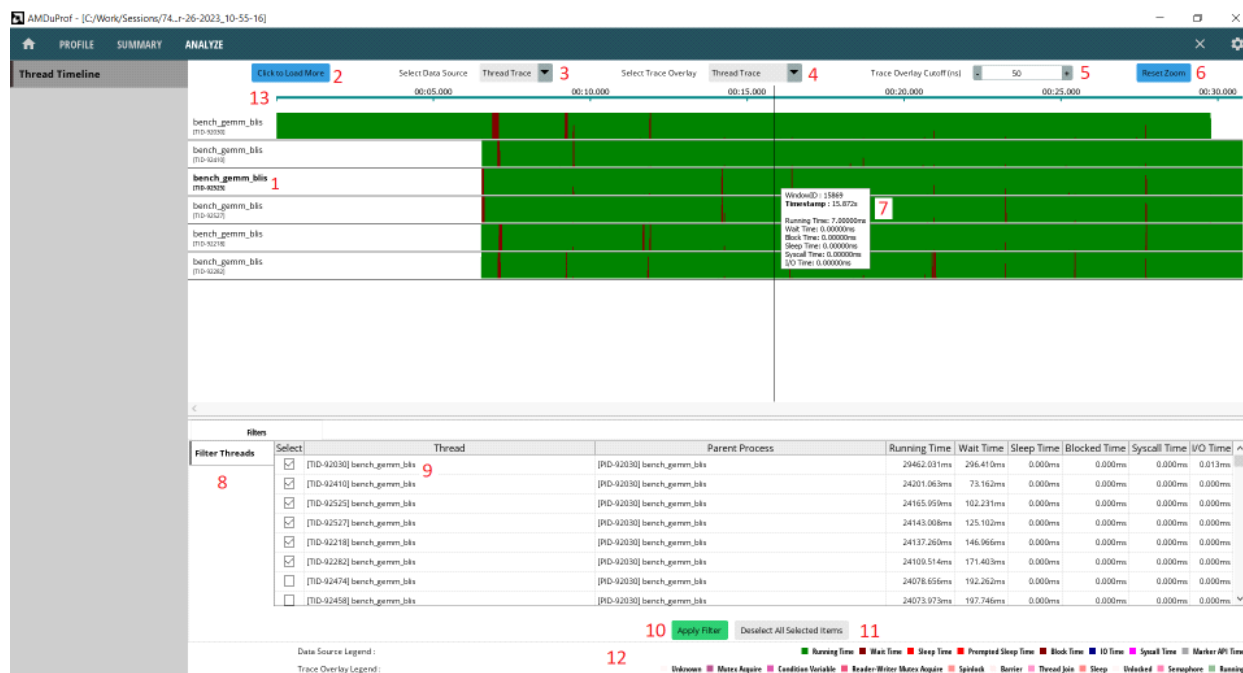


図 51. Linux のタイムライン解析 GUI

タイムライン セクションの構成は次のとおりです。

- タイムラインに含まれる各スレッドの名前とスレッド ID。
- 読み込むスレッドを増やすための **[Click to Load More]** ボタン。デフォルトでは、リソース消費を制限するため、少数のスレッド タイムラインのみが読み込まれています。このボタンをクリックすると、次のスレッド タイムライン セットが読み込めるようになります。次のセットは、タイムラインの下にあるテーブル内のエンティティによって決まります。

3. **[Data Source]** ドロップダウンを選択すると、タイムラインに表示するデータを選択できます。データソースの種類は次のとおりです。
- **[CPU Utilization]:** スレッドあたりの CPU 使用率 (%) のタイムラインが 1 秒間隔でプロットされます。十分なデータ ポイントを収集するには、総プロファイル時間を 10 秒以上にする必要があります。これは、**[Threading Analysis]** 設定のみで有効です。
 - **[Memory Consumption]:** 消費された物理メモリと仮想メモリに分けて、メモリ消費 (MB) のタイムラインがプロットされます。これは、**[Threading Analysis]** 設定のみで有効です。
 - **[Context Switches]:** 自発的コンテキスト スイッチ カウント (スリープ、イールドなど) または非自発的コンテキスト スイッチ カウント (OS スケジューラにトリガーされるコンテキスト スイッチ) のタイムラインがプロットされます。これは、**[Threading Analysis]** 設定のみで有効です。
 - **[CPU Profile Samples]:** CPU イベントに対して収集された CPU サンプルのタイムラインがプロットされます。サポートされるイベントは次のとおりです。

表 43. サポートされる CPU イベント

イベント	利用可能性
Retired Instructions	PMC イベント RETIRED_INSTRUCTIONS が収集されている。
Cycles not in Halt	PMC イベント CYCLES_NOT_IN_HALT が収集されている。
Op Cycles	IBS オペレーション イベントが「カウント サイクル」ユニット マスクと共に収集されている。
CPU Time	時間ベース プロファイリングが実行されている。

- **[Thread Trace]:** eBPF トレースまたはユーザー モード トレースのいずれかから収集された OS トレース データに基づくタイムラインがプロットされます。トレース データが分類され、一定の間隔で集計されてから、タイムラインにプロットされる時系列が生成されます。作成されるカテゴリは次のとおりです。

表 44. CPU トレース カテゴリ

カテゴリ	説明
待機時間	同期オブジェクト (ミューテックス、条件変数、セマフォ、ロック、バリア、ラッチなど) に消費された合計時間
スリープ時間	スリープ システム コールに消費された合計時間。
実行時間	ユーザー モード トレースのみが有効になっている場合は、次のとおりです。 実行時間 = 合計時間 – (待機時間 + スリープ時間)。 eBPF トレースが有効になっている場合、実行時間は CPU の合計アクティブ時間です。 実行時間 = 合計時間 – スリープ時間 (コンテキスト スイッチ レコードから)
ブロック時間	システム コール (select、epoll、poll、wait、accept など) のブロックに消費された合計時間。
I/O 時間	I/O システム コール (read、write、pread、pwrite など) に消費された合計時間。
システム コール 時間	トレースされるすべてのシステム コールで消費された合計時間 (ブロック時間 + I/O 時間)。

4. **[Select Trace Overlay]** ドロップダウンでは、表示するトレース データの種類を次から選択できます。
 - **[Don't Show Trace]:** トレース データはタイムラインに読み込まれません。
 - **[Thread State]:** eePBF またはユーザー モード トレースに含まれるスレッドの現在のステートが表示されます。eePBF では、スレッド ステートは BPF データから推定されます。ユーザー モード トレースでは、実行時間が 0 より大きい場合のスレッド ステートは**実行中**、それ以外の場合は**スリープ中**として処理されます。
 - **[Thread Trace]:** pthread_mutex_lock、pthread_mutex_trylock など、トレースされた libpthread 関数のトレースが表示されます。
 - **[Syscalls]:** 特定のタイムライン領域で、トレースされたシステム コールのトレースが表示されます。
5. **[Trace Cutoff]** を使用すると、時間をナノ秒 (ns) で指定できます。これはトレース データの読み込みをカットオフする役割を果たし、かかった時間が指定したナノ秒を下回るトレース対象関数は表示されません。
6. **[Reset Zoom]** ボタンをクリックすると、それまでに実行されたズームがすべてリセットされます。
7. タイムラインの任意の箇所にカーソルを合わせると、関連するデータとタイムスタンプを含むツール ヒントが表示されます。トレース データも存在する場合は、関連するトレース対象関数、開始時間、持続時間が表示されます。
8. **[Filter Threads/Ranks]** により、表示する必要のあるスレッド (またはランク) のタイムラインをフィルターできます。デフォルトでは、タイムラインが内部で並べ替えられ、最初の 6 件が読み込まれます。ただし、テーブルから必要なスレッドを選択して **[Apply Filter]** をクリックすると、変更を適用できます。CPU プロファイル データが収集されている場合、関数またはモジュールのハイライトも可能です。各関数にランダムな色が割り当てられ (変更可能)、タイムラインでハイライトされて、その関数/モジュールから取得されたサンプルがあることを意味します。
9. フィルター テーブル内のエントリごとに、必要なデータとして、名前、親オブジェクト、プロファイル全体で集計されたサンプル/トレース時間が含まれます。
10. **[Apply Filter]** ボタンをクリックすると、エンティティのカスタム選択またはタイムラインでのエンティティのハイライト表示が適用されます。
11. **[Deselect selected Items]** をクリックすると、フィルター テーブル内で最初のエントリを除くすべてのエントリの選択が解除されます。これは、カスタム選択が必要だが、すべてのタイムラインが既に読み込まれている場合に便利です。
12. フィルター パネルの下部にタイムラインの凡例が表示されており、「データ ソース」または「トレース」のタイプごとに割り当てられた色を識別できます。
13. **[Show Core Transition]** ボタンはデフォルトで無効になっており、CPU プロファイリング データが収集されている場合のみ機能します。このボタンが有効な場合、各タイムライン内に赤い線が表示され、スレッドでコアが変更されたタイミングが示されます。
14. CSS が有効な状態でプロファイリングされた設定がある場合は、**[Threading Analysis]** → **[Select Data Source]** → **[CPU Profile Samples]** を選択します。有効なサンプル領域を選択した場合のみ、コールスタック セクションが有効になります。

注記: ([Select Data Source] による) 時系列データは線グラフとしてプロットされます。ここで、x 軸は時間で、y 軸が高さになり、どれだけ最大値に近づいたかが示されます。トレースレコードの場合、高さは常にタイムラインの合計高さです。ただし、幅はトレースされる関数の持続時間によって異なります。

8.2 OpenMP の解析

OpenMP API は、並列実行の **fork-join** モデルを使用します。プログラムの開始時は、1 つのマスター スレッドがシリアル コードを実行します。並列領域になると、複数のスレッドが、**OpenMP** 指示子によって定義された暗示的または明示的なタスクを実行します。並列領域の終了時に、スレッドは境界で結合し、マスター スレッドのみが実行を継続します。

並列領域のコードを実行する際、スレッドは使用可能なすべての CPU コアを利用すべきであり、CPU 使用率を最大にする必要があります。ただし、スレッドは、次のような理由がある場合、有用な処理を実行せずに待機します。

- **アイドル:** あるスレッドが並列領域内でタスクを終了し、境界でその他のスレッドの完了を待機する。
- **同期:** 並列領域内でロックが使用されている場合、スレッドは同期ロックが共有リソースを獲得するまで待機できる。
- **オーバーヘッド:** スレッド管理のオーバーヘッド。

OpenMP 解析により、OpenMP スレッドによって実行されるアクティビティとそのステートをトレースできます。また、並列領域のスレッド ステート タイムラインを使用して、パフォーマンスの問題を解析できます。

サポート マトリックス

次の表に、サポート マトリックスを示します。

表 45. サポート マトリックス

コンポーネント	サポートされるバージョン	言語
OpenMP 仕様	OpenMP v5.0	
コンパイラ	LLVM 8、9、10、11、12、13、14	C および C++
	AOCC 2.1、2.2、2.3、3.0、3.1、3.2、4.0	C、C++、Fortran
	ICC 19.1 および 2021.1.1	C、C++、Fortran
OS	Ubuntu 18.04 LTS、20.04 LTS、22.04 LTS	
	RHEL 8.6 および 9	
	CentOS 8.4	

使用要件

サポートされるプラットフォームで、サポートされるコンパイラを使用し、**OpenMP** を有効化するのに必要なコンパイラ オプションを指定して、**OpenMP** アプリケーションをコンパイルします。

8.2.1 GUI を使用した OpenMP アプリケーションのプロファイリング

プロファイリングの設定と開始

OpenMP のプロファイリングを有効にするには、次の手順に従います。

1. プロファイル ターゲットとプロファイル タイプを選択します。
2. **[Advanced Options]** ボタンをクリックします。
3. 次に示すように、**[Enable OpenMP Tracing]** ペインで、**[Enable OpenMP Tracing]** オプションをオンにします。

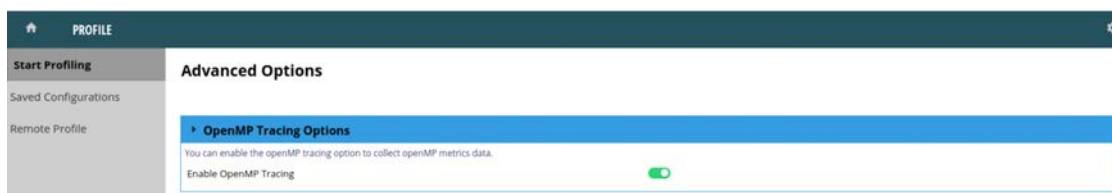


図 52. OpenMP トレースの有効化

OpenMP レポートの解析

プロファイリングの完了後、**[HPC]** ページに移動して、OpenMP トレース データを解析します。このページで左側の縦型ペインを使用して、次のビューに移動できます。

- **[Overview]** には、実行時間に関する概要が表示されます。次に、**[Overview]** ページを示します。

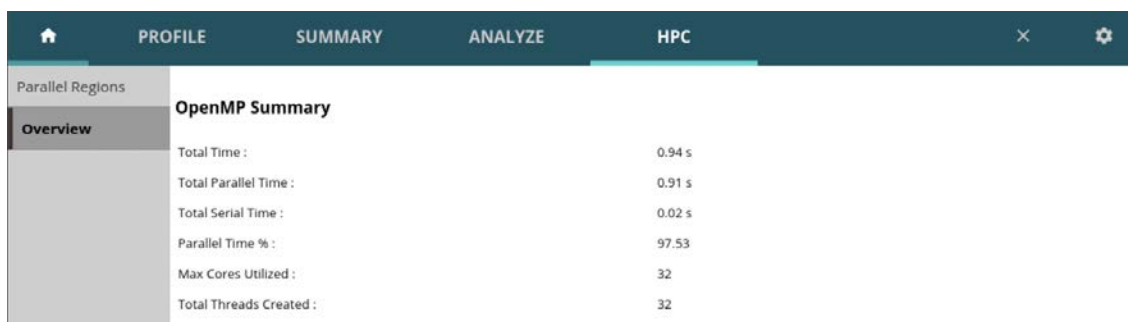


図 53. [HPC] - [Overview]

- **[Parallel Regions]** には、すべての並列領域のサマリが表示されます。このタブは、どの並列領域のロードが不均衡であるかをすばやく理解するのに役立ちます。領域名をダブルクリックすると、**[Regions Detailed Analysis]** ページが開きます。

Parallel Region	Imbalance Time	Imbalance Time	Threads	Avg Idle Time (s)	Avg Sync Time (s)	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::trans	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
CellPairTraj	0.000000	0.000000	256	0.000000	0.003310	0.000248	3.119357	1	Static	3.122916
LinkedCells::une	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

図 54. [HPC] - [Parallel Regions]

8.2.2 CLI を使用した OpenMP アプリケーションのプロファイリング

プロファイル データの収集

AMD uProf CLI を使用して OpenMP アプリケーションをプロファイリングするには、次のコマンドを使用します。

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

標準プロファイリングを実行する場合、`--trace openmp` または `--omp` オプションを追加して、OpenMP プロファイリングを有効にします。このコマンドによりプログラムが起動され、OpenMP 解析レポートの生成に必要なプロファイル データが収集されます。

OpenMP イベントのトレース モードは次のとおりです。

- フルトレース: すべての OpenMP イベントが、フル モードでトレースされます。フル OpenMP トレースを実行するには、次のコマンドを実行します。

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

- 基本トレース: 概要レポートの生成に必要なイベントのみがトレースされます。収集されるトレース データのサイズは、フルトレース モードよりも少なくなります。これがデフォルト モードです。基本 OpenMP トレースを実行するには、次のコマンドを実行します。

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

プロファイル レポートの生成

AMDuProfCLI report コマンドを使用して、CSV レポートを生成できます。OpenMP レポートの生成に、追加オプションは必要ありません。AMD uProf により、使用できる OpenMP プロファイリング データがあるかどうかチェックされ、使用できるデータがある場合はレポートに組み込まれます。

次のコマンドにより、CSV レポートが /tmp/myapp_perf/<SESSION-DIR>/report.csv に生成されます。

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

次に、CSV ファイル内の OpenMP レポート セクションの例を示します。

OpenMP TRACING REPORT (Time/durations are in seconds.)										
OpenMP OVERVIEW (PID-27842)										
Total Time		2.37								
Parallel Time		2.36								
Serial Time		0.01								
Parallel Time %		99.78								
Max cores utilized		6								
Total threads created		4								
OpenMP PARALLEL-REGION METRIC (PID-27842)										
Region	Imbalance Time	Imbalance Time(%)	Threads	Idle Time	Sync Time	Overhead	Work Time	Loop Chur	Schedule	Elapsed Time
collatz_sequence_compute\$omp\$parallel_for:4@collatz-sequence-omp-10pr.c:34	0.000007	0.001417	4	0.000007	0	0.025989	0.450394	1	Static	0.476391
collatz_sequence_compute\$omp\$parallel_for:4@collatz-sequence-omp-10pr.c:34	0.000005	0.001008	4	0.000005	0	0.023332	0.447906	1	Static	0.471243
collatz_sequence_compute\$omp\$parallel_for:4@collatz-sequence-omp-10pr.c:34	0.000006	0.001224	4	0.000006	0	0.023204	0.446558	1	Static	0.469768
collatz_sequence_compute\$omp\$parallel_for:4@collatz-sequence-omp-10pr.c:34	0.000009	0.001862	4	0.000009	0	0.0233	0.44654	1	Static	0.469849
collatz_sequence_compute\$omp\$parallel_for:4@collatz-sequence-omp-10pr.c:34	0.000239	0.050082	4	0.000239	0	0.021354	0.456124	1	Static	0.477718
OpenMP THREAD METRIC (collatz_sequence_compute\$omp\$parallel_for:4@collatz-sequence-omp-10pr.c:34)										
ThreadNum	Threadid	Idle Time		Sync Time	Overhead	Work Time				
0	27842		0	0	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				

図 55. OpenMP レポート

次に示すサブセクションが含まれます。

- **OpenMP OVERVIEW**
 - **OpenMP PARALLEL-REGION METRIC** により、不均衡な領域を把握できます。不均衡な領域とは、合計時間に対して合計処理時間が少ない領域です。次に示す列が含まれます。
 - **Imbalance Time:** 並列領域のすべてのスレッドによって消費された合計アイドル時間をスレッド数で正規化したもの。
 - **Imbalance Time (%):** 並列領域で消費された合計時間に対する不均衡な時間の割合。
 - **Threads:** 並列領域に含まれるスレッドの数。
 - **Avg Idle Time:** 並列領域のスレッドが、その他のスレッドの完了を境界で待機している間に消費した平均時間。
 - **Avg Sync Time:** 並列領域のスレッドが、共有リソースを獲得するための同期ロックを待機している間に消費した平均時間。
 - **Avg Overhead Time:** スレッド管理のオーバーヘッド。
 - **Avg Work Time:** 並列領域スレッドによる処理に消費された平均時間。
 - **Loop Chunk Size:** 特定のチャンクに対してスケジューリングされたループの回数。
 - **Schedule Type:** 関連付けられたループの繰り返しをチャンクに分割する方法と、スレッド間でこれらのチャンクを割り当てる方法を指定します。
 - **Elapsed Time:** 並列領域内で消費された時間。
 - **OpenMP THREAD METRIC** は、各スレッドが並列領域内でどのように時間を消費したかを理解するのに役立ちます。スレッドが処理アクティビティ以外に過度な時間を費やしている場合、並列領域の最適化を進めて、領域内の各スレッドの処理時間を改善する必要があります。次に示す列が含まれます。
 - **ThreadNum:** スレッドのシリアル番号。
 - **ThreadId:** スレッド識別子。
 - **Idle Time:** スレッドが、その他のスレッドの完了を境界で待機する間に消費した時間。
 - **Sync Time:** スレッドが、共有リソースを獲得するための同期ロックを待機する間に消費した時間。
 - **Overhead Time:** スレッド管理のオーバーヘッド。
 - **Work Time:** スレッドによる処理に消費された時間。
- OpenMP トレース データを Linux で収集してから、Windows 上の GUI または CLI にセッションをインポートできます。

8.2.3 環境変数

AMDUPROF_MAX_PR_INSTANCES - トレースする並列領域の最大数を設定します。デフォルト値は 2000 です。

8.2.4 制限

このリリースでは、次の機能はサポートされていません。

- プロファイリング スcopeがシステム全体である場合の OpenMP プロファイリング。
- `schedule` 節を使用してパラメーターが指定された場合の、ループ チャンク サイズとスケジュール タイプ。この場合は、デフォルト値 (1 と Static) が表示されます。
- ネストされた並列領域。
- GPU のオフロードと関連コンストラクト。
- 個別 OpenMP スレッドのコールスタック。
- Windows および FreeBSD プラットフォームでの OpenMP プロファイリング。
- OpenMP ライブラリの静的リンクを含むアプリケーション。
- 実行中の OpenMP アプリケーションへの接続。

8.3 MPI のプロファイリング

`mpirun` または `mpiexec` ランチャー プログラムで起動された MPI プログラムは、AMD uProf でプロファイリングできます。MPI アプリケーションをプロファイリングしてデータを解析するには、次の手順に従います。

1. CLI の `collect` コマンドを使用して、プロファイル データを収集します。
2. CLI の `translate` コマンドを使用して、プロファイル データを処理し、プロファイル データベースを生成します。
3. プロファイル データベースを GUI にインポートするか、CLI の `report` コマンドを使用して CSV レポートを生成します。
4. 複数ランクのプロファイリングでは、次のいずれかの方法を使用して、メモリ ロックの上限をより高く設定する必要があります。
 - ターゲット ノードでプロファイリングされるランク数に応じ、`ulimit -l` コマンドを使用してメモリ ロックの上限を大きくします。
 - プロファイルの設定とスコープに基づいて、`/proc/sys/kernel/perf_event_paranoid` を -1 以上の値に設定します。
 - ルート権限で MPI プロファイリングを実行します。
5. 複数ランクのプロファイリングでは、多数のファイル ディスクリプターが必要になる場合があります。プロファイル データ収集中にファイル ディスクリプターの上限に達すると、エラー メッセージが表示されます。`/etc/security/limits.conf` ファイル内に指定されているこの上限を増加できます。
6. 複数ランクのプロファイリングで、`/proc/sys/kernel/perf_event_paranoid` の値が -1 より大きい場合、プロファイリングするランク数に応じて、`/proc/sys/kernel/perf_event_mlockb` の値を増やす必要があります。別の方法として、`-m` オプションを使用して、AMDuProfCLI の各インスタンスが使用するメモリ データバッファー ページ数を小さくすることもできます。

サポート マトリックス

MPI プロファイリングは、次のコンポーネントと該当バージョンをサポートしています。

表 46. MPI プロファイリングのサポート マトリックス

コンポーネント	サポートされるバージョン
MPI 仕様	MPI v3.1
MPI ライブラリ	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、22.04 LTS
	RHEL 8.6 および 9
	CentOS 8

8.3.1 CLI を使用したデータの収集

MPI ジョブは、*mpirun* や *mpiexec* などの MPI ランチャーを使用して起動します。MPI アプリケーション向けのプロファイルデータを収集するには、AMDuProfCLI を使用する必要があります。

mpirun を使用して MPI ジョブを起動する場合、次の構文を使用します。

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

<program>を使用して AMDuProfCLI を起動し、AMDuProfCLI の引数を使用してアプリケーションを起動します。AMDuProfCLI を使用して MPI アプリケーションをプロファイリングするには、次の構文を使用します。

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

MPI プロファイリングに固有の AMDuProfCLI オプションは、次のとおりです。

- `--mpi` オプションを使用すると、MPI アプリケーションをプロファイリングできます。AMDuProfCLI によって、追加のメタ データが MPI プロセスから収集されます。
- `--output-dir <output dir>` を使用して、プロファイル ファイルが保存されるディレクトリのパスを指定します。<output dir> 内に、すべてのランクから収集されたデータをすべて含むセッション ディレクトリが作成されます。

標準的なコマンドでは、次の構文を使用します。

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

MPI アプリケーションが複数のノードで起動している場合、AMDuProfCLI によって、すべてのノードで実行されているすべての MPI ランク プロセスがプロファイリングされます。データ解析の対象は、単一/複数/すべてのノードで実行されたプロセスから選択できます。

方法 1 - 単一/複数ノードでのすべてのランクのプロファイリング

単一ノードで実行中のすべてのランクに対してプロファイル データを収集するには、次のコマンドを実行します。

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

複数ノードですべてのランクに対してプロファイル データを収集するには、`-H / --host mpirun` オプションを使用するか、`-hostfile <hostfile>` を指定します。

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

方法 2 - 特定ランクのプロファイリング

host2 で実行中の単一ランクのみをプロファイリングするには、次のコマンドを実行します。

```
$ 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
```

2つのホストで256のランクが実行されている場合(ホストあたり128のランク)に単一ランクのみをプロファイリングするには、次のコマンドを実行します。

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

方法 3 - MPI 設定ファイルの使用

`mpirun` に入力として設定ファイルを指定することもでき、`AMDuProfCLI` で設定ファイルを使用して MPI アプリケーションをプロファイリングできます。

設定ファイル (`myapp_config`) を使用するには、次のコマンドを実行します。

```
#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
```

この設定を使用して、host2 で実行中の MPI プロセスのデータのみを収集するには、次のコマンドを実行します。

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

8.3.2 CLI を使用したデータの解析

収集された MPI プロセスのデータは、`AMDuProfCLI` の `report` コマンドによって生成される CSV を使用して解析できます。生成されるレポートは、`<output-dir>/<SESSION-DIR>` フォルダの `file report.csv` 内に保存されます。

CLI のレポート オプションは、次のとおりです。

- MPI ランチャーが起動したローカル ホスト (例: host1) で実行されたすべての MPI プロセスに対してレポートを生成するには、次のコマンドを実行します。 `--input-dir`:

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

ローカル ホスト用のレポート ファイルを生成する場合、`--host` オプションは必須ではありません。

- MPI ランチャーが起動しなかった別のホスト (例: host2) で実行されたすべての MPI プロセスに対してレポートを生成するには、次のコマンドを実行します。

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

- すべてのホストで実行されたすべての MPI プロセスに対してレポートを生成するには、次のコマンドを実行します。

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

8.3.3 GUI を使用したデータの解析

GUI でプロファイル データを解析するには、次の手順に従います。

1. プロファイル データベースの生成については、[165 ページの「CLI を使用したデータの解析」](#)を参照してください。
2. プロファイル データベースのインポートについては、[72 ページの「プロファイル データベースのインポート」](#)を参照してください。

8.3.4 制限

Total number of ranks や **Number of ranks running on each node** などの MPI 環境パラメーターは、現在、OpenMPI のみでサポートされています。プロファイリング スコープがシステム全体である場合の MPI プロファイリングはサポートされていません。

8.4 Linux での perf_event_paranoid 値に対するプロファイリングサポート

次の表に、Linux での各種の perf_event_paranoid 値に対するプロファイリング サポートを示します。

表 47. Linux での perf_event_paranoid 値のプロファイリング

設定	プロファイル スコープ	perf_event_paranoid の値			
		-1	0	1	2
時間ベース プロファイリング	特定のアプリケーションまたはプロセス	Y	Y	Y	Y
時間ベース プロファイリング	カーネル、ハイパーバイザー	Y	Y	Y	N
時間ベース プロファイリング	システム全体	Y	Y	N	N
コア PMC イベント ベースのプロファイリング	特定のアプリケーションまたはプロセス	Y	Y	Y	Y
コア PMC イベント ベースのプロファイリング	カーネル、ハイパーバイザー	Y	Y	Y	N
コア PMC イベント ベースのプロファイリング	システム全体	Y	Y	N	N
命令ベースのサンプリング	特定のアプリケーションまたはプロセス	Y	Y	N	N
命令ベースのサンプリング	システム全体	Y	Y	N	N

8.5 Linux システム モジュールのプロファイリング

サンプルの原因となるシステム モジュール (例: glibc、libm) を特定するために、AMD uProf では、対応するデバッグ情報ファイルが使用されます。Linux ディストリビューションにはデバッグ情報ファイルが含まれていませんが、一般的なディストリビューションのほとんどで、デバッグ情報ファイルをダウンロードするためのオプションが提供されています。

デバッグ情報ファイルのダウンロード方法の詳細は、次に示すリソースを参照してください。

- 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)

プロファイリングの開始前に、目的のシステム モジュールおよび Linux ディストリビューション用のデバッグ情報ファイルを必ずダウンロードします。

8.6 Linux カーネルのプロファイリング

Linux カーネルのモジュールおよび関数をプロファイリングして解析するには、次の手順に従います。

1. カーネル シンボルの解決を有効にします。
2. 次のいずれかの手順に進みます。
 - カーネルのデバッグ シンボル パッケージおよびソースをダウンロードしてインストールします。
 - デバッグ シンボルを含む Linux カーネルを構築します。

デフォルト パス内でカーネル デバッグ情報を使用できるようになったら、AMD uProf は自動的にデバッグ情報を見つけて利用し、ソース ビューにカーネルのソース行とアセンブリを表示します。

サポートされる OS: Ubuntu 18.04 LTS、Ubuntu 20.04 LTS、RHEL 7、RHEL 8

8.6.1 カーネル シンボルの解決の有効化

カーネル サンプルの原因となる適切なカーネル関数を特定するため、AMD uProf は `/proc/kallsyms` ファイルから必要な情報を抽出します。`/proc/kallsyms` を介してカーネル シンボル アドレスを公開するには、次のように、`/proc/sys/kernel/kptr_restrict` ファイルに適切な値を設定する必要があります。

- `/proc/sys/kernel/perf_event_paranoid` を「-1」に設定します。
- `/proc/sys/kernel/kptr_restrict` を、次に示す適切な値に設定します。
 - **0:** カーネル アドレスを制限なしで使用できます。
 - **1:** 現在のユーザーが CAP_SYSLOG 機能を有する場合、カーネル アドレスを使用できます。
 - **2:** カーネル アドレスは使用できません。

次のいずれかを使用して、`perf_event_paranoid` の値を設定します。

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

または

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

次のいずれかを使用して、**kptr_restrict** の値を設定します。

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

または

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

8.6.2 カーネル デバッグ シンボル パッケージのダウンロードとインストール

Linux システムでは、**/boot** ディレクトリ内に、圧縮された **vmlinux** イメージか、圧縮されていない **vmlinux** イメージが含まれています。このようなカーネル ファイルには、シンボルおよびデバッグ情報は含まれていません。デバッグ情報がない場合、AMD uProf はサンプルの原因となるカーネル関数を特定できません。このため、デフォルトでは、AMD uProf はカーネル関数をレポートしません。

一部の Linux ディストリビューションでは、カーネル向けのデバッグ シンボル ファイルが提供されているため、これをプロファイリング目的に使用できます。

Ubuntu

Ubuntu システムにカーネル デバッグ情報とソース コードをダウンロードするには、次の手順に従います (Ubuntu 18.04.03 LTS で検証済み)。

1. デバッグ シンボルの署名キーを信頼するには、次のコマンドを実行します。

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

2. 次の方法で、デバッグ シンボル リポジトリを追加します。

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

3. 使用可能なデバッグ シンボル パッケージの一覧を取得します。

```
$ sudo apt update
```

4. 現在のカーネル バージョンのデバッグ シンボルをインストールします。

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

5. カーネル ソースをダウンロードします。

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

または

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

次に示すように、カーネル デバッグ情報ファイルがダウンロードされると、デフォルト パスに格納されます。

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

RHEL

RHEL カーネル デバッグ情報をダウンロードするには、Red Hat ナレッジベース (<https://access.redhat.com/solutions/9907>) に記載された手順に従います。

次に示すように、カーネル デバッグ情報ファイルがダウンロードされると、デフォルト パスに格納されます。

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

8.6.3 デバッグ シンボルを含む Linux カーネルの構築

構築済みのカーネル イメージで利用できるデバッグ シンボル パッケージがない場合、ソース レベルでカーネル関数を解析するには、デバッグ フラグを有効にして Linux カーネルを再コンパイルする必要があります。

8.6.4 カーネル関数内のホットスポットの解析

カーネル モジュールのデバッグ情報が使用可能になると、その後のすべての CPU パフォーマンス解析で、カーネル スペース サンプルの原因となる **[vmlinux]** モジュールが適切に特定され、ホット カーネル関数が表示されます。それ以外の場合、カーネル サンプルの原因として **[kernel.kallsyms]_text** モジュールが示されます。

ホットスポット解析の実行中は、次に示す点を考慮してください。

- **[vmlinux]** モジュールが表示される場合、GUI のソース ビューと IMIX ビューでカーネル関数のパフォーマンス データを解析できます。CLI を使用する場合も、そのカーネルのソース レベル レポートと IMIX レポートを生成できます。
- ソースがダウンロードされて所定のパスにコピーされている場合、GUI と CLI でカーネル ソース行を表示できます。
- パフォーマンスの問題につながる場合があるため、カーネル デバッグ ファイルのパスとカーネル ソースのパスを渡すことは推奨されません。

8.6.5 Linux カーネルのコールスタック サンプリング

システム全体のプロファイリングで、カーネル関数のコールスタック サンプルを収集できます。たとえば、次のコマンドにより、カーネル コールスタックを収集できます。

```
# 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 制約

- ダウンロードされたカーネル デバッグ情報を、デフォルト パスから移動しないでください。
- カーネル バージョンがアップグレードされた場合は、最新のカーネル バージョンに対応するカーネル デバッグ情報をダウンロードします。カーネル デバッグ情報とカーネル バージョンの間で何らかの不一致がある場合、AMD uProf で正しいソースとアセンブリは表示されません。
- カーネル サンプルのプロファイリングまたは解析中は、合間にシステムをリブートしないでください。システムをリブートすると、Linux カーネルの KASLR 機能により、異なる仮想アドレスがカーネルにロードされます。

- `/proc/sys/kernel/kptr_restrict` ファイル内の設定により、AMD uProf はカーネル シンボルを解決し、サンプルの原因となるカーネル関数を特定できるようになります。ソースおよびアセンブリ レベルのコールグラフ解析は有効になりません。

8.7 カーネル ブロック I/O 解析

`insert`、`issue`、`complete` のような Linux OS のブロック I/O 呼び出しをトレースすると、アプリケーションによって実行された I/O オペレーションに関連する各種メトリクスを提供できます。

表 48. I/O オペレーション

カテゴリ	イベント	説明
OS およびランタイム	diskio	アプリケーション実行中のブロック I/O オペレーションをトレースします。

この解析を使用して、次に示す要素を解析できます。

- I/O オペレーションの完了までにかかった時間
- IOPS - 1 秒あたりのブロック I/O オペレーションの数
- ブロック I/O オペレーションの読み出しまたは書き込みバイト数
- ブロック I/O 帯域幅

注記: カーネルは、アプリケーションの終了後も引き続き、プロファイル対象アプリケーションから送信され、キューに追加された I/O 要求を実行できます。このため、この解析ではシステム全体のトレースを使用することを推奨します。

使用要件

OS イベントとランタイム ライブラリをトレースするための条件は、次のとおりです。

- Linux kernel 4.7 以降 (kernel 4.15 以降を推奨)。
- Linux で OS イベントをトレースするには、ルート アクセスが必要です。
- BCC と eBPF スクリプトのインストールについては、[6 ページの「BCC のインストールと eBPF」](#)を参照してください。BCC インストールを検証するには、`sudo AMDuProfVerifyBpfInstallation.sh` スクリプトを実行します。

8.7.1 CLI を使用したカーネル ブロック I/O 解析

AMDuProfCLI を使用して、必要なトレース データを収集し、詳しい分析のために `.csv` 形式でレポートを生成できます。また、処理済みのプロファイル データを GUI にインポートできます。

プロファイル データの収集

次に、ブロック I/O オペレーションを時間ベース サンプリングと共にトレースするための CLI コマンド例を示します。

```
$ sudo AMDuProfCLI collect --config tbp -trace os=diskio -o /tmp/blockio-analysis/ /usr/bin/fio ...
...
Generated data files path: /tmp/blockio-analysis/AMDuProf-fio-0sTrace_Dec-09-2021_12-19-27
```

このコマンドを使用すると、プログラムが起動して、プロファイルおよびトレース データを収集します。起動アプリケーションを実行すると、AMDuProfCLI はセッション ディレクトリ パスを表示します。このディレクトリ内に、生のプロファイルおよびトレース データが保存されています。

上に示した例で、セッション ディレクトリ パスは次のとおりです。

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

プロファイル レポートの生成

次の CLI report コマンドを使用し、セッション ディレクトリ パスをオプション -i の引数として渡して、.csv 形式でプロファイル レポートを生成します。

```
$ ./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
```

データを処理してレポートを生成すると、ターミナルにレポート ファイル パスが表示されます。次に、.csv レポート ファイル内のディスク I/O レポート セクションの例を示します。

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		

図 56. ディスク I/O サマリ テーブル

GUI を使用したトレース データの解析

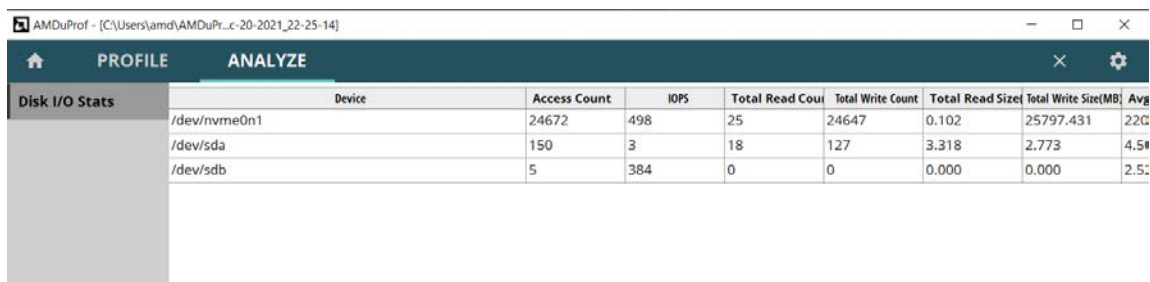
CLI を使用して収集されたトレース データを視覚化するには、GUI にインポートする前に、収集された生プロファイルおよびトレース データを CLI translate コマンドを使用して処理する必要があります。

次に示す CLI translate コマンドの呼び出しを使用して、対応するセッション ディレクトリ パスに保存された生トレース レコードを処理します。

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

次に、このセッションを GUI にインポートするため、[HOME] → [Import Session] ビューで、[Profile Data File] テキスト入力ボックスにセッション ディレクトリ パスを指定します。これにより、セッション ディレクトリに保存されたプロファイル データが今後の解析用に読み込まれます。

次のように、[ANALYZE] ページに移動して、縦型のナビゲーション バーで [Disk I/O Stats] を選択します。



Device	Access Count	IOPS	Total Read Count	Total Write Count	Total Read Size(MB)	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	4.5*
/dev/sdb	5	384	0	0	0.000	0.000	2.5*

図 57. 解析 - ブロック I/O 統計

上の図のテーブルに、デバイス レベルの各種ブロック I/O 統計情報が示されています。

8.8 GPU オフロード解析 (GPU トレース)

GPU オフロード解析は、GPU 計算量の多いアプリケーションでの関数呼び出しのトレースを調査するために使用できます。

AMD ROCtracer ライブラリは、データ転送やカーネル実行など、ランタイム API と GPU アクティビティを取り込むためのサポートを提供します。この解析は、HIP ベースのアプリケーションの実行中に、ROCr、HIP API 呼び出し、GPU アクティビティを視覚化するのに役立ちます。これは、起動アプリケーションのみでサポートされます。

サポートされるインターフェイス

AMD uProf では、次の ROCr ランタイム API、GPU アクティビティのトレースがサポートされており、GUI タイムラインビューにこれらのデータが表示されます。

表 49. GPU トレースがサポートされるインターフェイス

カテゴリ	イベント	説明
GPU	hip	HIP ランタイムのトレース
GPU	hsa	AMD ROCr ランタイムのトレース

使用要件

ROCr、HIP API、GPU アクティビティをトレースするための条件は、次のとおりです。

- AMD ROCm 5.5 がインストールされている必要があります。AMD ROCm のインストール手順については、[6 ページの「ROCm のインストール」](#)を参照してください。

注記: 「5.2.1 以前」のバージョンでは、トレースが想定どおりに動作しない場合があります。

- サポート アクセラレータ - AMD Instinct™ MI100 および MI200

オプション設定

デフォルトで、AMDuProf は次を使用します。

- `/opt/rocm/` シンボリック リンクによって参照される ROCm バージョン。rocm パスを指定するには、AMD uProf を起動する前に、AMDUPROF_ROCM_PATH を使用してこれをエクスポートする必要があります。

例:

```
export AMDUPROF_ROCM_PATH=/opt/rocm-5.5.0/
```

- `/opt/rocm/lib` に含まれる ROCm ライブラリ。AMDUPROF_ROCM_PATH を指定すると、指定されたパスまたはライブラリが使用されます。このパスを変更するには、AMD uProf を起動する前に、AMDUPROF_ROCM_LIB_PATH を使用してこれをエクスポートする必要があります。

例:

```
export AMDUPROF_ROCM_LIB_PATH=/opt/rocm-5.5.0/lib
```

8.8.1 CLI を使用した GPU オフロード解析

AMDuProfCLI を使用して、必要なトレース データを収集し、詳しい分析のために .csv 形式でレポートを生成できます。また、処理済みのプロファイル データを GUI にインポートできます。

プロファイル データの収集

CLI の `--trace` オプションを使用すると、トレースする GPU イベントとランタイム ライブラリを指定できます。HIP ベースのアプリケーションで、GPU オフロード解析を実行するために、時間ベースのサンプリングと共に ROCr、HIP API、GPU アクティビティをトレースするための CLI コマンド例は、次のとおりです。

```
$ 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
```

このコマンドを使用すると、プログラムが起動して、プロファイルおよびトレース データを収集します。起動アプリケーションを実行すると、AMDuProfCLI はセッション ディレクトリ パスを表示します。このディレクトリ内に、生のプロファイルおよびトレース データが保存されています。

上に示した例で、セッション ディレクトリ パスは次のとおりです。

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

GPU プロファイル収集が割り込まれた場合、または起動アプリケーションが別のターミナルから中止された場合の動作は不定です。

プロファイル レポートの生成

次の CLI `report` コマンドを使用し、セッション ディレクトリ パスをオプション `-i` の引数として渡して、.csv 形式でプロファイル レポートを生成します。

```
$ ./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
```

データを処理してレポートを生成すると、ターミナルにレポート ファイル パスが表示されます。次に、.csv レポート ファイル内の GPU トレース レポート セクションの例を示します。

GPU TRACING REPORT				
KERNEL SUMMARY				
Name	Count	Elapsed Time(second)	Avg Elapsed Time	Elapsed Time(% From Total API Elapsed Time)
JacobilerationKernel(int, double, dc	1000	0.549017	0.000549017	41.9465
NormKernel1(int, double, double, do	1001	0.470506	0.000470036	35.948
LocalLaplacianKernel(int, int, int, dou	1000	0.270641	0.000270641	20.6777
HaloLaplacianKernel(int, int, int, dou	1000	0.015341	1.53E-05	1.1721
NormKernel2(int, double const*, dou	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

図 58. GPU トレース レポート

GUI からの GPU トレースの詳細は、7.8.1 を参照してください。

8.9 GPU のプロファイリング

AMD ROCprofiler ライブラリが提供するサポートにより、GPU カーネルがディスパッチされて実行される際の、GPU ハードウェア パフォーマンス イベントを監視できます。これらのイベントに基づくパフォーマンス メトリクスは算出されてから、CSV レポート内に記載されます。これは、起動アプリケーションのみでサポートされます。

使用要件

GPU パフォーマンス プロファイリングの条件は、次のとおりです。

- AMD ROCm 5.5 がインストールされている必要があります。AMD ROCm のインストール手順については、[6 ページの「ROCm のインストール」](#)を参照してください。

注記: 「5.2.1 以前」のバージョンでは、プロファイリングが想定どおりに動作しない場合があります。

- サポートされるアクセラレータ - AMD Instinct™ MI100 および MI200

サポートされるイベントとメトリクス

サポートされる GPU パフォーマンス メトリクスは、次のとおりです。ターゲット システム上のサポート対象イベント一覧を表示するには、AMDuProfCLI `info --list gpu-events` コマンドを実行します。

次の表に、サポートされるイベントの一覧を示します。

表 50. GPU プロファイリングでサポートされるイベント

イベント	説明
GRBM_COUNT	GPU フリーランニング クロック
GRBM_GUI_ACTIVE	GPU ビジー クロック
SQ_WAVES	SQ に送信されたウェーブのカウンタ数。(simd あたり、エミュレート対象、グローバル)
TCC_HIT_sum	キャッシュ ヒットの数。
TCC_MISS_sum	キャッシュ ミスの数。UC 読み出しはミスとしてカウント。
SQ_INSTS_VALU	発行された VALU 命令の数。(simd あたり、エミュレート対象)
SQ_INSTS_SALU	発行された SALU 命令の数。(simd あたり、エミュレート対象)
SQ_INSTS_SMEM	発行された SMEM 命令の数 (simd あたり、エミュレート対象)
SQ_INSTS_LDS	発行された LDS 命令の数 (FLAT を含む) (simd あたり、エミュレート対象)
SQ_INSTS_GDS	発行された GDS 命令の数 (simd あたり、エミュレート対象)
TCC_EA_RDREQ_sum	TCC/EA 読み出し要求の数 (32 バイトまたは 64 バイト)
TCC_EA_RDREQ_32B_sum	32 バイト TCC/EA 読み出し要求の数
SQ_ACTIVE_INST_VALU	VALU 命令に対して SQ 命令アービタが動作しているサイクルの数 (simd あたり、非確定的)
SQ_THREAD_CYCLES_VALU	VALU オペレーションの実行に使用されたスレッド サイクル数 (simd あたり)

表 50. GPU プロファイリングでサポートされるイベント (続き)

イベント	説明
TA_FLAT_READ_WAVEFRONTS_sum	TA によって処理された flat オペコード読み出しの数
TA_FLAT_WRITE_WAVEFRONTS_sum	TA によって処理された flat オペコード書き込みの数

次の表に、サポートされるメトリクスの一覧を示します。

表 51. GPU プロファイリングでサポートされるメトリクス

メトリクス	説明
GPU_UTIL (%)	GPU 使用率 (%)
VALU_UTIL (%)	VALU 使用率 (%)
VALU_THREAD_DIVERGENCE (%)	VALU スレッドの平均逸脱率 (%)
L2_CACHE_HIT_RATE (%)	平均 L2 キャッシュ ヒット率 (%)
VALU_INSTR (IPW)	ウェーブあたりの平均 VALU 命令数
SALU_INSTR (IPW)	ウェーブあたりの平均 SALU 命令数
SMEM_INSTR (IPW)	ウェーブあたりの平均 SMEM 命令数
LDS_INSTR (IPW)	ウェーブあたりの平均 LDS 命令数
GDS_INSTR (IPW)	ウェーブあたりの平均 GDS 命令数
L2_CACHE_HITS (PW)	ウェーブあたりの平均 L2 キャッシュ ヒット数
L2_CACHE_MISSES (PW)	ウェーブあたりの平均 L2 キャッシュ ミス数
EA_32B_READ (PW)	ウェーブあたりの平均 32 バイト読み出し数
EA_64B_READ (PW)	ウェーブあたりの平均 64 バイト読み出し数
EA_READ_BW (GB/s)	1 秒あたりの読み出し帯域幅 (GB)

8.9.1 CLI を使用した GPU プロファイリング

プロファイル データの収集

GPU パフォーマンス データを収集するには、次のコマンドを使用します。

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

このコマンドを使用すると、プログラムが起動して、プロファイル データを収集します。起動アプリケーションを実行すると、AMDuProfCLI はセッション ディレクトリ パスを表示します。このディレクトリ内に、生のプロファイル データが保存されています。

上に示した例で、セッション ディレクトリ パスは次のとおりです。

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

GPU プロファイル収集が割り込まれた場合、または起動アプリケーションが別のターミナルから中止された場合の動作は不定です。

プロファイルレポートの生成

次の CLI report コマンドを使用し、セッションディレクトリパスをオプション `-i` の引数として渡して、`.csv` 形式でプロファイルレポートを生成します。

```
$ ./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
```

データを処理してレポートを生成すると、ターミナルにレポートファイルパスが表示されます。次に、`.csv` レポートファイル内の GPU プロファイルレポート セクションの例を示します。

GPU PROFILE REPORT											
KERNEL STATS											
Name	Count	Elapsed Time	Avg Elapsed Time	Elapsed Time	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_COUNT	GRBM_GUI_AC	SQ_WAVES	TCC_HIT_s	TCC_MISS	SQ_INSTS	SQ_INSTS	SQ_INSTS	SQ_INSTS	SQ_INSTS	
__amd_rocclr_fillBuffer.kd	655979072	655940864	9127359	8.4E+08	1.4E+08	3.05E+09	9.57E+08	76825480	3.71E+08	0	
void bondedForcesKernel<	334267520	334267520	1257630	3.64E+08	45374256	1.44E+09	4.51E+08	11863202	1.92E+08	0	
void nonbondedForceKern	250795680	250795680	1218813	1.81E+09	84609696	1.52E+10	1.67E+09	38583192	8.44E+08	0	
void nonbondedForceKern	223318368	223318368	912751	1.35E+09	42918508	1.48E+10	2.85E+09	42899296	1.03E+09	0	
void modifiedExclusionFor	220224080	220224080	1102476	1.48E+09	76276864	1.21E+10	1.32E+09	30845012	6.89E+08	0	
void nonbondedForceKern	200330336	200330336	1105181	1.63E+09	59872448	1.59E+10	1.49E+09	36560968	1.04E+09	0	
void scalar_sum_kernel<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 Al	Max LDS A	Min LDS Al	Avg Scrata	
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	

図 59. GPU プロファイルレポート

8.10 その他の OS トレース イベント

170 ページの「カーネルブロック I/O 解析」に記載された OS イベント以外に、次の OS イベントを CPU サンプリング ベース プロファイルと共にトレースできます。

表 52. OS トレースでサポートされるイベント

イベント	説明
pagefault	ページ フォルト数をトレースします。
memtrace	メモリの割り当ておよび割り当て解除呼び出しをトレースします。デフォルトでは、1 KB 以上のメモリ割り当てのみがトレースされます。 注記: これは、アプリケーションレベルのトレースのみでサポートされます。
funccount	--func オプションで指定された関数をトレースします。

使用要件

OS イベントとランタイム ライブラリをトレースするための条件は、次のとおりです。

- Linux kernel 4.7 以降 (kernel 4.15 以降を推奨)。
- Linux で OS イベントをトレースするには、ルート アクセスが必要です。
- BCC と eBPF スクリプトのインストールについては、6 ページの「BCC のインストールと eBPF」を参照してください。BCC インストールを検証するには、`sudo AMDuProfVerifyBpfInstallation.sh` スクリプトを実行します。

8.10.1 CLI を使用したページ フォルトとメモリ割り当てのトレース

AMDuProfCLI を使用して、必要なトレース データを収集し、詳しい分析のために .csv 形式でレポートを生成できます。

プロファイル データの収集

CLI の --trace オプションを使用すると、トレースする OS イベントとランタイム ライブラリを指定できます。パフォーマンスを包括的に解析する目的で、時間ベース サンプリングと共に、ページ フォルトおよびメモリ割り当てをトレースする CLI コマンド例は、次のとおりです。

```
$ 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
```

このコマンドを使用すると、プログラムが起動して、プロファイルおよびトレース データを収集します。起動アプリケーションを実行すると、AMDuProfCLI はセッションディレクトリパスを表示します。このディレクトリ内に、生のプロファイルおよびトレース データが保存されています。

上に示した例で、セッションディレクトリパスは次のとおりです。

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

次の CLI report コマンドを使用し、セッションディレクトリパスをオプション -i の引数として渡して、.csv 形式でプロファイル レポートを生成します。

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

データを処理してレポートを生成すると、ターミナルにレポート ファイル パスが表示されます。次に、.csv レポート ファイル内の GPU トレース レポート セクションの例を示します。

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

図 60. ページ フォルトおよびメモリ割り当てのサマリ

8.10.2 CLI を使用した関数呼び出しカウントのトレース

OS トレースでの funccount は、特定のモジュール (実行ファイル/ライブラリまたはカーネル関数) の関数をカウントします。1 回にトレースできる関数の最大数は、1000 です。

CLI のオプションについては、[88 ページの表 27](#) を参照してください。

次に、.csv レポート ファイル内の関数カウント レポート セクションの例を示します。

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

図 61. 関数カウントのサマリ

例:

- AMDTClassicMatMul-bin によって libc から呼び出された malloc() 関数のカウントを収集します。libc は、デフォルト ライブラリ パス内で検索されます。

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

- コンテキスト スイッチ、syscall、pthread API トレース、AMDTClassicMatMul-bin によって呼び出された malloc() の関数カウントを収集します。

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

- malloc()、calloc()、カーネル関数のうち、パターン「vfs_read*」に一致する関数カウントをシステム全体で収集します。

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

- AMDTClassicMatMul-bin からのすべての関数のカウントを収集します。

```
$ AMDuProfCLI collect --trace os=funccount --func /home/amd/AMDTClassicMatMul-bin:-* -o /tmp/cpuprof-os AMDTClassicMatMul-bin
```

GUI からの GPU トレースの詳細は、7.8.1 を参照してください。

8.11 MPI トレース解析

MPI トレース解析を使用して、同一クラスターで実行される MPI アプリケーションのランク間でのメッセージパス ロードの不均衡を解析および算出できます。サポートされるのは、OpenMPI、MPICH、およびこれらの派生プログラムです。

サポートされるスレッド モデルは、SINGLE、FUNNLED、SERIALIZED です。生成されるプロファイル レポートには、ポイント ツー ポイントと集合型 API アクティビティ サマリがあります。

Fortran バインドは、MPI インプリメンテーションのコンパイル中に設定および構築されます。Fortran 言語のサポートが必要かどうかに基づいて、Fortran バインドを有効または無効にできます。

Fortran バインドを有効または無効にするには、次に示すオプションを参照してください。

- OpenMPI

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

デフォルトで、OpenMPI は 3 つの Fortran バインド (mpif.h、mpi モジュール、mpi_f08 モジュール) をすべて構築しようと試行します。

- MPICH

```
--disable-fortran
```

デフォルトで、Fortran バインドは有効になっています。このオプションを使用すると無効にできます。

サポート マトリックス

表 53. サポート マトリックス

コンポーネント	サポートされるバージョン
MPI 仕様	MPI v3.1
MPI ライブラリ	Open MPI v4.1.4、MPICH v4.0.3、ParaStation MPI v5.6.0、Intel® MPI 2021.1
OS	• Ubuntu: 18.04 LTS、20.04 LTS、22.04.04 LTS • RHEL: 8.6 および 9 • CentOS 8.4
言語	C、C++、Fortran

トレース モード

AMDuProf CLI では、次の 2 つの MPI トレース モードをサポートしています。

- **LWT** – 軽量トレースは、素早いアプリケーション解析に有用です。レポートは、収集ステージの実行中に、**.csv** 形式で生成されます。
- **FULL** – フルトレースは、詳細な解析に有用です。このモードでは、**.csv** 形式でのレポート生成のために処理後の作業が必要です。

MPI インプリメンテーションのサポート

AMD uProf では、Open MPI および MPICH と、これらの派生プログラムのトレースがサポートされています。

- `--trace mpi=mpich`: MPICH とその派生プログラム向け (デフォルト オプション)
- `--trace mpi=openmpi`: Open MPI 向け

MPI アプリケーションのコンパイルに使用された MPI インプリメンテーションに合わせて、必ず正しいオプション (`mpich` または `openmpi`) を渡します。渡されたオプションが正しくないと、動作が不定になる場合があります。

MPI トレース オプションの詳細は、[88 ページの「Linux 専用オプション」](#)を参照してください。

8.11.1 CLI を使用した MPI 軽量トレース

LWT モードでは、収集ステージでクイック レポートが生成されます。このモードでサポートされるトレース対象 API は限られています。このレポートは、次に示すアプリケーション ランタイム アクティビティの概要を提供します。

表 54. 軽量トレースでサポートされる MPI API

MPI_Bsend	MPI_Recv_init	MPI_Bcast	MPI_Ireduce_scatter
MPI_Bsend_Init	MPI_Rsend	MPI_Gather	MPI_Iscan
MPI_Ibsend	MPI_Rsend_init	MPI_Gatherv	MPI_Iscatter
MPI_Improbe	MPI_Send	MPI_Iallgather	MPI_Iscatterv
MPI_Imrecv	MPI_Send_init	MPI_Iallgatherv	MPI_reduce
MPI_Iprobe	MPI_Ssend	MPI_Iallreduce	MPI_reduce_scatter
MPI_Irecv	MPI_Ssend_Init	MPI_Ialltoall	MPI_Scan
MPI_Irsend	MPI_Allgather	MPI_Ialltoallv	MPI_Scatter
MPI_Isend	MPI_Allgatherv	MPI_Ialltoallw	MPI_Scatterv
MPI_Issend	MPI_Allreduce	MPI_Ibarrier	MPI_Wait
MPI_Mprobe	MPI_Alltoall	MPI_Ibcast	MPI_Waitall
MPI_Mrecv	MPI_Alltoallv	MPI_Igather	MPI_Waitany
MPI_Probe	MPI_Alltoallw	MPI_Igatherv	MPI_Waitsome
MPI_Recv	MPI_Barrier	MPI_Ireduce	

プロファイル データの収集

AMDuProfCLI を使用して、MPI アプリケーションを LWT するコマンド例は、次のとおりです。

```
$ mpirun -np <number of processes> ./AMDuProfCLI collect --trace mpi=lwt -o <output_directory>
<application>
```

トレースの完了後、セッション ディレクトリへのパスがターミナルに表示されます。LWT レポートは、収集完了後すぐに生成され、次のようにセッション ディレクトリ内に保存されます。<output_directory>/<SESSION_DIR>/mpi/lwt/mpi-summary.csv。

MPI インプリメンテーションとして、MPICH または Open MPI をコマンド内に指定する必要があります。デフォルトは MPICH です。

次にサンプル コマンドを示します。

```
$ 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>
```

MPI アプリケーションのコンパイルに使用された MPI インプリメンテーションに合わせて、必ず正しいオプション (mpich または openmpi) を渡します。渡されたオプションが正しくないと、動作が不定になる場合があります。

次に、.csv ファイル内の LWT レポート セクションの例を示します。

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	

図 62. LWT レポート

8.11.2 CLI を使用した MPI フルトレース

フルトレース モードでは、LWT よりも多くの API がトレースされます。このモードは、MPI アプリケーションのアクティビティを詳しく解析するために役立ちます。

フルトレースのレポート ファイルには、さまざまな情報を示す複数のテーブルが含まれます。

- コミュニケーター サマリには、次の列が含まれます。
 - **Communicator Size:** メンバー ランクの数
 - **Elapsed Time:** コミュニケーター内で MPI API によって消費された時間。
 - **Ranks:** メンバー ランク ID
- ランク サマリには、次の列が含まれます。
 - **Rank:** ランク ID。
 - **PID:** プロセス ID。
 - **MPI Time (seconds):** MPI API に対して消費された合計時間。
 - **MPI Time (%):** すべてのランクの合計 MPI 時間に対する MPI 時間の割合。
 - **Wait Time (seconds):** ランクの待機により消費された時間。
 - **Wait Time (%):** アプリケーション実行時間に対するランク待機時間の割合。
 - **Call Count:** MPI API が呼び出された回数。
 - **Volume (bytes):** 送信または受信したデータ量 (バイト)。
 - **Volume (%):** すべてのランクが送信または受信した合計データ量に対するデータ量の割合。
 - **Elapsed Time (seconds):** アプリケーション実行時間。
 - **Time (%):** 合計経過時間に対する経過時間の割合。
- P2P API サマリには、次の列が含まれます。
 - **Function:** MPI API の名前。
 - **Min Time (seconds):** すべてのランクでこの API に費やされた合計時間の最小時間。
 - **Max Time (seconds):** すべてのランクでこの API に費やされた合計時間の最大時間。
 - **Average Time (seconds):** この API に費やされた平均時間。
 - **MPI Time (%):** すべての MPI API に費やされた合計時間に対するこの API に費やされた時間の割合。
 - **Volume (Bytes):** この MPI API が送信または受信した合計データ量。
 - **Calls:** この MPI API が呼び出された回数。
 - **Total Time (seconds):** すべてのランクでこの API に費やされた合計時間。

- コミュニケーター マトリックスには、次の列が含まれます。
 - **Rank:** 送信ランク ID と受信ランク ID。
 - **MPI Time (seconds):** 送信ランクから受信ランクにデータを送信する API に費やされた合計時間。
 - **MPI Time (%):** すべての API に費やされた合計 MPI 時間に対する MPI 時間の割合。
 - **Volume (Bytes):** 送信ランクから受信ランクに送信された合計データ量。
 - **Volume (%):** すべてのランク間で転送された合計データ量に対するデータ量の割合。
 - **Transfers:** 送信ランクから受信ランクへの転送回数。
- 集合型 API サマリには、次の列が含まれます。
 - **Function:** API の名前。
 - **Min Time (seconds):** この API に費やされた最小時間。
 - **Max Time (seconds):** この API に費やされた最大時間。
 - **Average time (seconds):** この API に費やされた平均時間。
 - **MPI Time (%):** すべての MPI 呼び出しに費やされた合計時間に対してこの API に費やされた時間の割合。
 - **Input Volume (Bytes):** この API 呼び出しに含まれるすべてのランクが受信した合計データ量 (バイト)。
 - **Output Volume (Bytes):** この API 呼び出しに含まれるすべてのランクが送信した合計データ量 (バイト)。
 - **Calls:** この API が呼び出された回数。
 - **Total Time (seconds):** すべてのランクでこの API に費やされた合計時間。

サポートされる MPI API の一覧は、次のとおりです。

表 55. MPI API

MPI_Pcontrol	MPI_Mrecv	MPI_Reduce	MPI_Iallreduce
MPI_Cancel	MPI_Imrecv	MPI_Allreduce	MPI_Ialltoall
MPI_Probe	MPI_Send	MPI_Alltoall	MPI_Ialltoallv
MPI_Iprobe	MPI_Bsend	MPI_Alltoallv	MPI_Ialltoallw
MPI_Mprobe	MPI_Ssend	MPI_Alltoallw	MPI_Ineighbor_Alltoall
MPI_Improbe	MPI_Rsend	MPI_Neighbor_Alltoall	MPI_Ineighbor_Alltoallw
MPI_Start	MPI_Bsend_init	MPI_Neighbor_Alltoallw	MPI_Ineighbor_Alltoallv
MPI_Startall	MPI_Ssend_init	MPI_Neighbor_Alltoallv	MPI_Ibarrier
MPI_Test	MPI_Rsend_init	MPI_Bcast	MPI_Ibcast
MPI_Testall	MPI_Send_init	MPI_Scan	MPI_Comm_create
MPI_Testany	MPI_Ibsend	MPI_Reduce_Scatter	MPI_Comm_dup
MPI_Testsome	MPI_Issend	MPI_Ireduce_Scatter	MPI_Comm_dup_with_info
MPI_Wait	MPI_Irsend	MPI_Iscan	MPI_Comm_split
MPI_Waitall	MPI_Isend	MPI_Iscatter	MPI_Comm_split_type
MPI_Waitany	MPI_Scatter	MPI_Iscatterv	MPI_Intercomm_create
MPI_Waitsome	MPI_Scatterv	MPI_Igather	MPI_Intercomm_merge
MPI_Barrier	MPI_Gather	MPI_Igatherv	MPI_Cart_create

表 55. MPI API (続き)

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	

プロファイルデータの収集

AMD uProf CLI を使用して、MPI アプリケーションを FULL トレースするコマンド例は、次のとおりです。

```
$ mpirun -np <number of processes> ./AMDuProfCLI collect --trace mpi=full -o <output_directory> <application>
```

トレースの完了後、セッション ディレクトリへのパスがターミナルに表示されます。

MPI インプリメンテーションとして、MPICH または Open MPI をコマンド内に指定する必要があります。デフォルトは MPICH です。

次にサンプル コマンドを示します。

```
$ 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>
```

MPI アプリケーションのコンパイルに使用された MPI インプリメンテーションに合わせて、必ず正しいオプション (mpich または openmpi) を渡します。渡されたオプションが正しくないと、動作が不定になる場合があります。

プロファイルレポートの生成

.csv 形式でレポートを生成するコマンド例は、次のとおりです。セッション ディレクトリ パスを -i オプションで指定します。

```
$ ./AMDuProfCLI report -i <output_directory>/<SESSION_DIR>
```

レポート生成が完了すると、*report.csv* ファイルのパスが端末に表示されます。

レポート ファイル内のテーブル

次のスクリーンショットに、フルトレース レポート ファイルの各セクションの例を示します。

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	

図 63. MPI コミュニケーター サマリ テーブル

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	

図 64. MPI ランク サマリ テーブル

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

図 65. MPI API サマリ テーブル

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	2540496160	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

図 66. MPI コミュニケーション マトリックス

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	0.0555552	6.1	53243309	7606187	64	3.55554
MPI_Reduce	1.13E-05	0.177126	0.0309416	1.27	144	1008	24	0.742599

図 67. MPI 集合型 API サマリ テーブル

8.11.3 GUI を使用した MPI フルトレース

トレースの収集とインポート

ターゲット MPI アプリケーションのトレースとレポートの生成には、CLI を使用します。手順については、[183 ページの「CLI を使用した MPI フルトレース」](#)を参照してください。次の図に示すように、レポートを GUI にインポートして、トレース データを解析します。

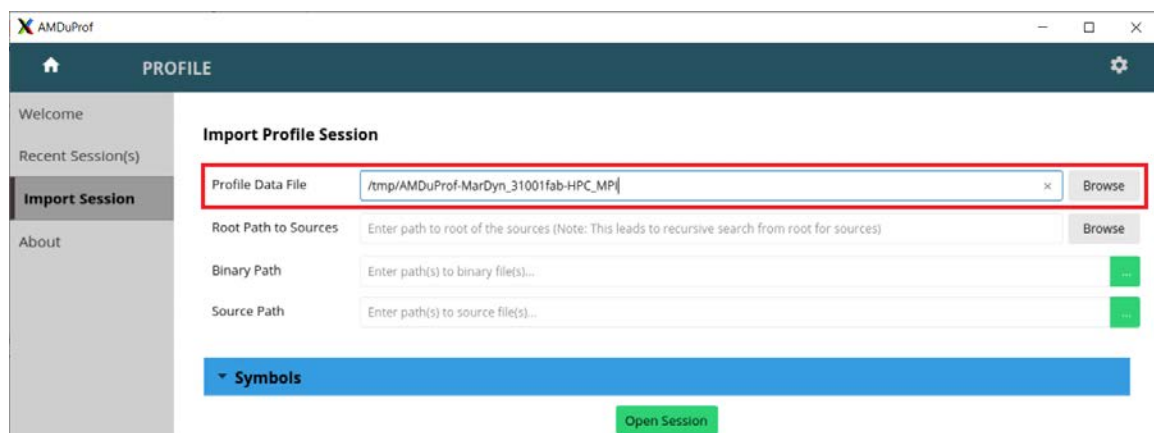


図 68. プロファイル セッションのインポート

MPI コミュニケーション マトリックスの解析

インポートが完了したら、[MPI Communication Matrix] ビューを使用し、GUI で MPI トレース データを解析します。[HPC] → [MPI Communication Matrix] に移動し、視覚化された MPI コミュニケーション マトリックスを表示します。このビューでは、ランク間のコミュニケーション サマリがマトリックス形式で表示されます。マトリックスの x 軸と y 軸は、それぞれ、受信ランクと送信ランクです。

次の図に、MPI コミュニケーション マトリックスを示します。



図 69. MPI コミュニケーション マトリックス

上の図の構成は次のとおりです。

1. 列方向と行方向に配列されたランク。
2. 各セルに、特定のランクから別のランクへと転送された合計データ量が表示されています。
3. マウスをセルに合わせると、ツール ヒントに詳細が表示されます。

- データ量に基づく色分けの凡例。
- そのランクに対するすべてのデータ転送の合計。
- そのランクに対するすべてのデータ転送の平均値。

MPI ランク タイムラインの解析

[HPC] → [MPI Rank Timeline] に移動し、MPI ランク タイムラインを表示します。このビューでは、次のように、MPI アクティビティがタイムライン グラフで表示されます。

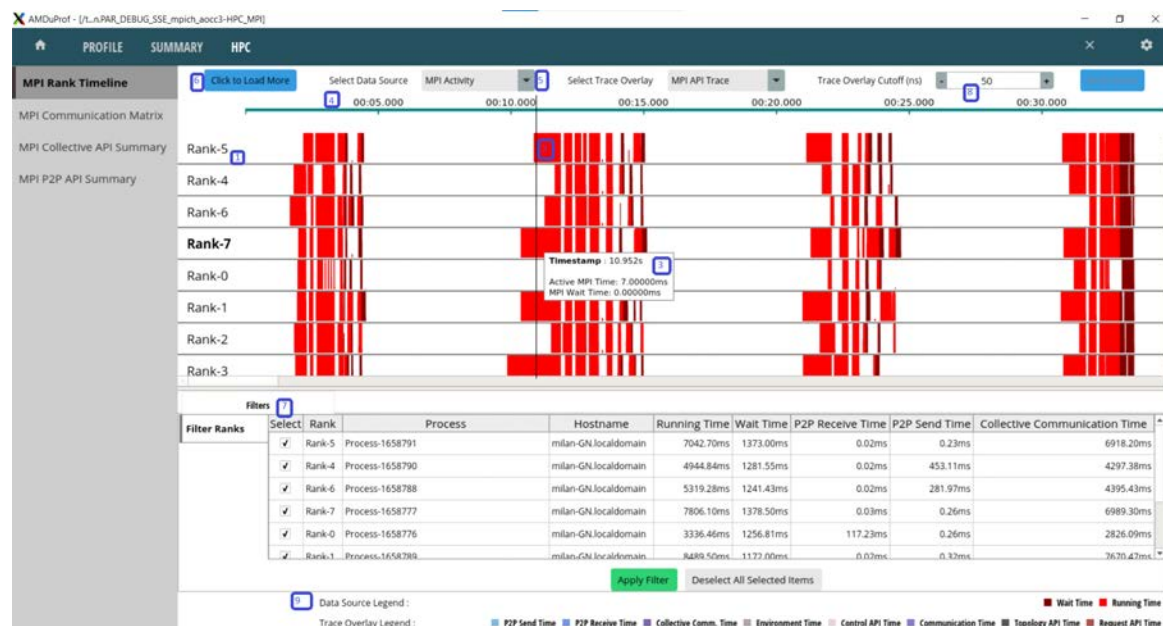


図 70. MPI ランク タイムライン

上のスクリーンショットの構成は、次のとおりです。

- ランク ID。
- 選択されたデータ ソースに応じて、次のいずれかのグラフ。
 - MPI API アクティビティ (実行中または待機中)
 - MPI データ転送アクティビティ (受信または送信)
 - MPI API 呼び出し
- MPI アクティビティに関する追加情報を表示するツール ヒント。
- 時間範囲の表示。
- データ ソースとして MPI アクティビティを選択します。詳細は、「MPI データ ソース」を参照してください。
- 読み込むランク情報を増やします。
- ビューに表示するランクをフィルターします。

8. **[Trace Overlay Cutoff]** を使用すると、時間をナノ秒 (ns) で指定できます。これはトレース データの読み込みをカットオフする役割を果たし、かかった時間が指定したナノ秒を下回るトレース対象データ ソースは表示されません。
9. データ ソースとトレース オーバーレイの色分け凡例。

MPI P2P API サマリの解析

[HPC] → [MPI P2P API Summary] に移動します。このビューは、次に示すように、アプリケーションによって呼び出された P2P API を要約したものです。

Function	Min Time	Max Time	Average Time	MPI Time(%) ▼	Volume(Bytes)	Calls
MPI Rank Timeline						
MPI_Cart_create	0.00s	0.58s	0.19s	5.29	0	16
MPI Communication Matrix						
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_Iprobe	0.00s	0.11s	0.00s	3.00	0	29276
MPI_Isend	0.00s	0.00s	0.00s	0.00	25433040	72
MPI_Irecv	0.00s	0.00s	0.00s	0.00	25433040	72

図 71. [MPI P2P API Summary]

MPI 集合型 API サマリの解析

[HPC] → [MPI PCollective API Summary] に移動します。このビューは、次に示すように、アプリケーションによって呼び出された集合型 API を要約したものです。

Function	Min Time	Max Time	Average Time	MPI Time(%) ▼	Input Volume(Bytes)	Output Volume(Bytes)	Calls
MPI Rank Timeline							
MPI_Allreduce	0.00s	1.98s	0.22s	62.27	2656	2656	168
MPI Communication Matrix							
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
MPI P2P API Summary							

図 72. [MPI Collective API Summary]

MPI データ ソース

サポートされる MPI データ ソースの一覧は次のとおりです。

- MPI アクティビティでは、MPI API が「待機中」API (MPI_Barrier、MPI_Wait、MPI_Waitall、MPI_Waitany、MPI_Waitsome) または「アクティブ」API (その他すべての MPI 関数) に分類されます。

- MPI API は、次のように分類されます。

P2P 送信	P2P 受信	集成型コミュニケーション
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

制御 API	要求 API	通信 API
MPI_PCONTROL	MPI_CANCEL MPI_START MPI_STARTALL MPI_TEST MPI_TESTALL MPI_TESTANY MPI_TESTSOME MPI_WAIT MPI_WAITALL MPI_WAITANY MPI_WAIT SOME 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

トポロジ API	環境 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 データ転送は、MPI P2P 送信/受信に分類され、その転送データ量が特定の時間間隔でプロットされます。

第 9 章 消費電力プロファイル

9.1 概要

システム全体の消費電力プロファイル

AMD uProf プロファイラーが提供するライブ消費電力プロファイリングにより、AMD CPU および APU ベースのシステムの動作を監視できます。消費電力特性と熱特性を監視するための各種カウンターが提供されます。

これらのカウンターは、RAPL や MSR など、さまざまなリソースから収集されます。カウンターは定期的な間隔で収集され、テキスト ファイルとしてレポートされるか、線グラフとしてプロットされます。また、将来的な解析のためにデータベース内にも保存できます。

機能

AMD uProf に含まれる機能は、次のとおりです。

- GUI を使用して、サポートされる消費電力メトリクスを設定および監視できます。
- TIMECHART ページでは、次を監視して解析できます。
 - 論理コア レベルのメトリクス - コア有効周波数と P ステート
 - 物理コア レベルのメトリクス - RAPL ベースのコア消費電力
 - パッケージ レベルのメトリクス - RAPL ベースのパッケージ消費電力と温度
- AMDuProfCLI timechart コマンドはシステム メトリクスを収集し、テキスト ファイルまたはカンマ区切り値 (CSV) ファイルに書き込みます。
- API ライブラリにより、サポートされるシステム レベルのパフォーマンスと、AMD CPU/APU の熱および消費電力メトリクスを設定し、収集できます。
- 収集されたライブ プロファイル データは、将来的な解析のためにデータベースに保存できます。

9.2 メトリクス

サポートされるメトリクスは、プロセッサ ファミリとモデルによって異なり、各種のカテゴリに大まかにグループ化されます。次に、プロセッサ ファミリ別のサポート対象カウンター カテゴリを示します。

表 56. Family 17h Model 00h ~ 0Fh (AMD Ryzen™、AMD Ryzen ThreadRipper™、第 1 世代 AMD EPYC™)

消費電力カウンター カテゴリ	説明
消費電力	サンプリング期間の平均消費電力をワットで示します。これは、プラットフォームのアクティビティ レベルに基づく推定消費値です。対象はコアとパッケージです。
周波数	サンプリング期間の CPU コア有効周波数を MHz で示します。
温度	サンプリング期間の平均温度を摂氏で示します。レポートされる温度は、Tctl の値です。対象はパッケージです。
P ステート	サンプリングが実行されたときの、CPU P ステート。

表 57. Family 17h Model 10h ~ 1Fh (AMD Ryzen™、AMD Ryzen™ PRO APU)

消費電力カウンター カテゴリ	説明
消費電力	サンプリング期間の平均消費電力をワットで示します。これは、プラットフォームのアクティビティ レベルに基づく推定消費値です。対象はコアとパッケージです。
周波数	サンプリング期間の CPU コア有効周波数を MHz で示します。
温度	サンプリング期間の平均温度を摂氏で示します。レポートされる温度は、Tctl の値です。対象はパッケージです。
P ステート	サンプリングが実行されたときの、CPU P ステート。

表 58. Family 17h Model 70h ~ 7Fh (第 3 世代 AMD Ryzen™)

消費電力カウンター カテゴリ	説明
消費電力	サンプリング期間の平均消費電力をワットで示します。これは、プラットフォームのアクティビティ レベルに基づく推定消費値です。対象はコアとパッケージです。
周波数	サンプリング期間の CPU コア有効周波数を MHz で示します。
P ステート	サンプリングが実行されたときの、CPU P ステート。
温度	サンプリング期間の平均温度を摂氏で示します。レポートされる温度は、Tctl の値です。対象はパッケージです。

表 59. Family 17h Model 30h ~ 3Fh (EPYC 7002)

消費電力カウンター カテゴリ	説明
消費電力	サンプリング期間の平均消費電力をワットで示します。これは、プラットフォームのアクティビティ レベルに基づく推定消費値です。対象はコアとパッケージです。
周波数	サンプリング期間の CPU コア有効周波数を MHz で示します。
P ステート	サンプリングが実行されたときの、CPU P ステート。
温度	サンプリング期間の平均温度を摂氏で示します。レポートされる温度は、Tctl の値です。対象はパッケージです。

表 60. Family 19h Model 0h ~ 2Fh (EPYC 7003、EPYC 9000)

消費電力カウンター カテゴリ	説明
消費電力	サンプリング期間の平均消費電力をワットで示します。これは、プラットフォームのアクティビティ レベルに基づく推定消費値です。対象はコアとパッケージです。
周波数	サンプリング期間の CPU コア有効周波数を MHz で示します。
P ステート	サンプリングが実行されたときの、CPU P ステート。
温度	サンプリング期間の平均温度を摂氏で示します。レポートされる温度は、Tctl の値です。対象はパッケージです。

9.3 GUI でのプロファイルの使用

システム全体の消費電力プロファイル (ライブ)

このプロファイル タイプは、メトリクスがライブ タイムライン グラフにプロットされているか、データベース内に保存されている場合、またはその両方である場合に、消費電力解析を実行するために使用できます。プロファイリングを設定して開始するには、次の手順に従います。

9.3.1 プロファイルの設定

プロファイリングを設定するには、次の手順に従います。

1. 上部のナビゲーション バーで **[PROFILE]** タブをクリックするか、**[Welcome]** ページで次のいずれかをクリックします。
 - **[Profile entire System]**
 - **[See What's guzzling power in your system]****[Select Profile Target]** ページが表示されます。
2. **[Next]** をクリックします。
 [Select Profile Type] ページが表示されます。
3. **[Select Profile Configuration]** 画面で、**[Live Power Profile]** タブを選択します。

次に示すように、すべてのライブ プロファイリング オプションと使用可能なカウンターがそれぞれのペイン内に表示されます。

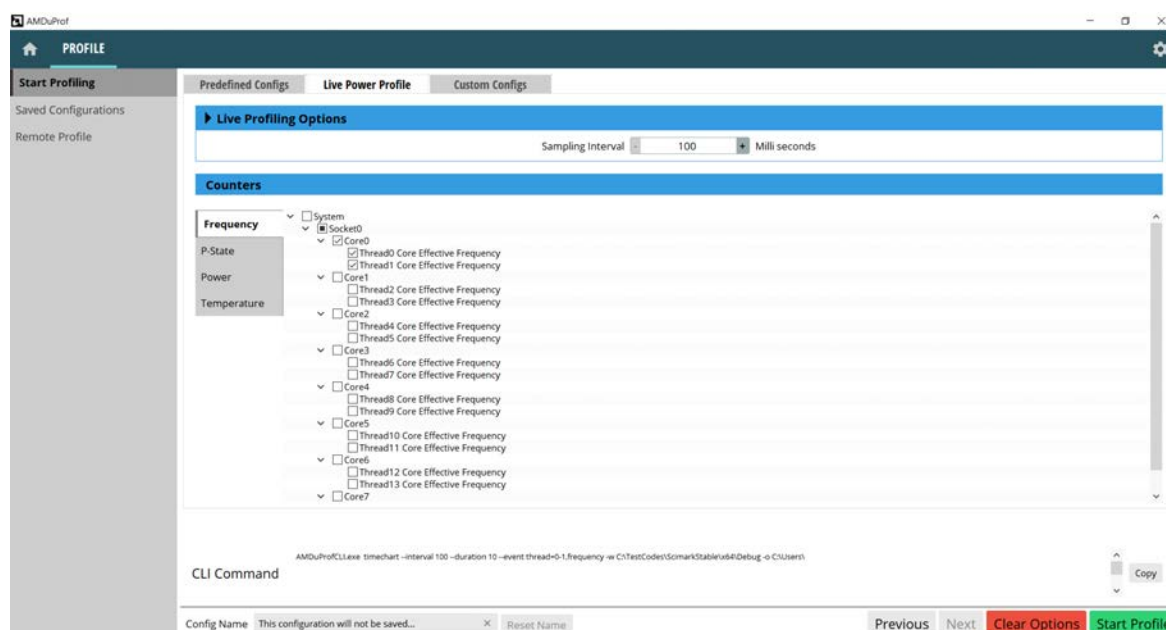


図 73. システム全体のライブ消費電力プロファイル

4. [Counters] ペインで、目的のカウンター カテゴリとそれぞれのオプションを選択します。

注記: 複数のカウンター カテゴリを設定できます。

プロファイリング中、ライブでグラフをレンダリングできます。

5. [Start Profile] をクリックします。

このプロファイル タイプでは、プロファイル データは [TIMECHART] ページで線グラフとして生成され、今後の解析で使用できます。

GUI でライブ消費電力プロファイリング用に選択されたすべてのオプションに対して、CLI コマンドが表示されます。

9.3.2 プロファイルの解析

必要なカウンターを選択し、プロファイル データの収集が開始されると、[TIMECHART] タブが開き、ライブ タイムライン グラフ内にメトリクスがプロットされます。



図 74. [Timechart] ページ

1. [TIMECHART] ページで、メトリクスはライブ タイムライン グラフ内にプロットされます。線グラフはグループにまとめられ、カテゴリに基づいてプロットされます。
2. 各グラフの横にあるデータ テーブルに、現在のカウンター値が表示されます。
3. [Graph Visibility] ペインで、表示するグラフを選択できます。

4. プロットの進行中に、次の操作を実行できます。

- **[Pause Graphs]** をクリックすると、データ収集を一時停止せずにグラフを一時停止できます。後で再開するには、**[Play Graphs]** をクリックします。
- **[Stop Profiling]** をクリックすると、ビューを閉じることなくプロファイリングを停止できます。この場合、プロファイルデータの収集は停止します。
- **[Close View]** をクリックすると、プロファイリングを停止してビューを終了できます。

9.4 CLI を使用したプロファイリング

AMDuProfCLI timechart コマンドを使用して、システム メトリクスを収集し、テキスト ファイルまたはカンマ区切り値 (CSV) ファイルに書き込むことができます。消費電力プロファイルのカウンター値を収集するには、次の手順に従います。

1. このコマンドに `--list` オプションを指定して実行すると、サポートされるカウンター カテゴリの一覧が表示されます。
2. このコマンドを使用し、`-e` または `--event` オプションで目的のカウンターを指定すると、必要なカウンターを収集してレポートできます。

次のように、timechart の実行により、サポートされるカウンター カテゴリのリストが表示されます。

```
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>
```

図 75. --list コマンドの出力

次のように、timechart の実行により、プロファイル サンプルが収集され、ファイルに書き込まれます。

```
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>
```

図 76. Timechart の実行

上記に示した実行により、対象カウンターがサポートされるすべてのデバイスで、消費電力と周波数のカウンターが収集され、-o オプションで指定された出力ファイルに書き込まれます。プロファイリングの開始前に、指定されたアプリケーションが起動して、アプリケーションが終了するまでの間、データを収集します。

9.4.1 例

Windows

- 100 ミリ秒のサンプリング間隔で 10 秒間にわたり、すべての消費電力カウンター値を収集します。

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

- 10 秒間にわたりすべての周波数カウンター値を収集し、500 ミリ秒ごとにサンプリングし、結果を csv ファイルに追加します。

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

- 10 秒間にわたり、コア 0 ~ 3 のすべての周波数カウンター値を収集し、500 ミリ秒ごとにサンプリングし、結果をテキスト ファイルに追加します。

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

Linux

- 100 ミリ秒のサンプリング間隔で 10 秒間にわたり、すべての消費電力カウンター値を収集します。

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

- 10 秒間にわたりすべての周波数カウンター値を収集し、500 ミリ秒ごとにサンプリングし、結果を csv ファイルに追加します。

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

- 10 秒間にわたり、コア 0 ~ 3 のすべての周波数カウンター値を収集し、500 ミリ秒ごとにサンプリングし、結果をテキスト ファイルに追加します。

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

9.5 AMDPowerProfileAPI ライブラリ

API ライブラリを使用すると、AMD uProf GUI または CLI を使用することなく、消費電力プロファイリング カウンターを各種の AMD プラットフォームで直接設定し、収集できます。AMDPowerProfileAPI ライブラリは、AMD CPU および APU ベースのシステムの電力効率を解析するために使用できます。

これらの API は、AMD CPU および APU とそのサブコンポーネントの消費電力、熱、周波数に関する特性を読み取るためのインターフェイスを提供します。これらの API のターゲットは、固有のユース ケースに基づいて、消費電力カウンターをサンプリングする独自アプリケーションを構築しようとするソフトウェア開発者です。

これらの API の詳細は、AMD uProf インストール フォルダー内にある *AMDPowerProfilerAPI.pdf* を参照してください。

9.5.1 API の使用

これらの API の使用法については、サンプル プログラム *CollectAllCounters.cpp* を参照してください。このプログラムでは、コンパイル時に *AMDPowerProfileAPI* ライブラリをリンクする必要があります。また、消費電力プロファイリング ドライバーのインストールと実行が必要です。

これらの API を使用するサンプル プログラム *CollectAllCounters.cpp* は、*<AMDuProf-install-dir>/Examples/CollectAllCounters/* ディレクトリにあります。サンプル アプリケーションをビルドして実行するには、使用している OS に合わせて次に示す手順を実行します。

Windows

Visual Studio 2015 のソリューション ファイル *CollectAllCounters.sln* が、*C:/Program Files/AMD/AMDuProf/Examples/CollectAllCounters/* ディレクトリにある状態で、サンプル プログラムをビルドします。

Linux

1. ビルドするには、次のコマンドを使用します。

```
$ cd <AMDuProf-install-dir>/Examples/CollectAllCounters
$ g++ -O -std=c++11 CollectAllCounters.cpp -I<AMDuProf-install-dir>/include -l A
MDPowerProfileAPI -L<AMDuProf-install-dir>/lib -Wl,-rpath <AMDuProf-install-dir>/bin -o
CollectAllCounters
```

2. 実行するには、次のコマンドを使用します。

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

9.6 制限

- 一度に実行できるのは、1 つの消費電力プロファイル セッションのみです。
- CLI でサポートされる最小サンプリング期間は、100 ms です。サンプリングとレンダリングのオーバーヘッドを低減するため、長いサンプリング期間を使用することを推奨します。

第 10 章 リモート プロファイリング

10.1 概要

AMD uProf には、リモート システムに接続して、リモート システム上のデータ収集および変換をトリガーし、ローカル GUI で視覚化する機能があります。

注記: CLI では、リモート プロファイリングはサポートされていません。

AMD uProf が使用する独立した AMDProfilerService バイナリは、リモート ターゲットでアプリケーション サーバーとして起動でき、ローカル GUI からこのサーバーに接続できます。デフォルトでは、ローカル GUI に接続するためにサーバー上に認可を設定する必要があります。次の手順を実行します。

1. ローカル GUI のクライアント ID を見つけます。
2. リモート ターゲットでクライアント ID を認可し、AMDProfilerService に接続します。
3. リモート ターゲットで、適切なオプション/権限を指定して AMDProfilerService を起動します。
4. ローカル GUI に接続情報を指定し、リモート ターゲットに接続します。
5. ローカル GUI が自動的に更新され、リモート データが表示されます (設定、セッション履歴、プロファイリング/トレースで使用可能なイベントなど)。
6. 続いてリモート ターゲットでセッション/プロファイルをインポートします。
7. リモート ターゲットでの作業が完了したら接続を切り、GUI でローカル データを更新します。

サポート

Windows (ホスト/ローカルプラットフォーム) から Linux (ターゲット/リモート プラットフォーム) へのリモート プロファイリングがサポートされています。

10.2 認可のセットアップ

認可をセットアップするには、次の手順に従います。

1. **[PROFILE]** → **[Remote Profile]** に移動し、**[Client ID]** を見つけます。

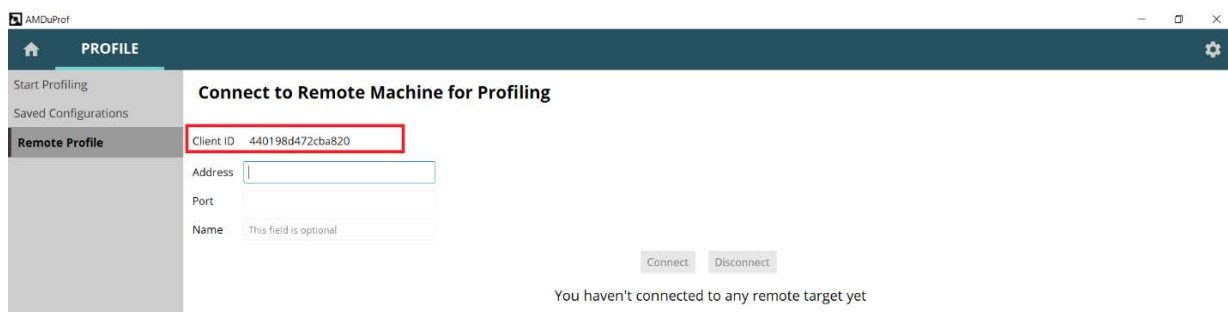


図 77. [Client ID]

2. **[Client ID]** (英数字) をコピーします。

3. リモート ターゲットで AMD uProf の bin ディレクトリに移動し、次のコマンドを実行します。

```
AMDProfilerService --add <client_id>
```

これにより、このリモート ターゲットへのクライアント接続が認可されます。

認可を取り消すには、次のコマンドを実行します。

```
AMDProfilerService --clear-user <client_id>
```

10.3 AMDProfilerService の起動

AMDProfilerService をアプリケーション サーバーとして起動するためのバインド IP アドレスを指定します。

```
AMDProfilerService --ip 127.0.0.1
```

この IP アドレスは、AMDProfilerService が起動されるターゲット/リモート マシンが持つ IP アドレスの 1 つである必要があります。

ターゲット/リモート マシンに IP アドレスが複数ある場合、ホスト/ローカル マシンで ping コマンドを使用すると、(リモート マシンの) どの IP アドレスにローカル マシンからアクセスできるかを判断できます。アクセス可能な IP アドレスを、--ip オプションで指定します。

(オプション) 指定できるオプションは次のとおりです。

表 61. AMDProfilerService のオプション

オプション	説明
--port <port_number>	ポート番号を指定します
--logpath <path>	ログ ファイルのパスを指定します
--bypass-auth	認可をスキップします 注記: このオプションにより認可が省略されるため、使用には注意が必要です。
--fsearch-depth <depth>	再帰的ファイル検索オペレーションの最大深さを指定します 注記: このオプションは、GUI からセッションをインポートする場合にのみ使用できます。
--fsearch-timeout <timeout>	再帰的ファイル検索オペレーションの最大時間 (秒) を指定します 注記: このオプションは、GUI からセッションをインポートする場合にのみ使用できます。

次に、リモート プロファイリングの接続確立画面の例を示します。

```
$ ./AMDProfilerService --ip 10.138.152.101 --port 32768
AMDuProf service started...
Listening for connection on port 32768 ...
```

図 78. リモート プロファイリングの接続の確立

次に、IP 選択画面の例を示します。

```
-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):
```

図 79. IP の選択

10.4 リモート ターゲットへの接続

リモート ターゲットに接続するには、次の手順に従います。

1. リモート ターゲットで AMDProfilerService が起動したら、次のように **[Remote Profile]** ページに移動して、IP アドレス、ポート番号、オプションのリモート ターゲット名を指定します。

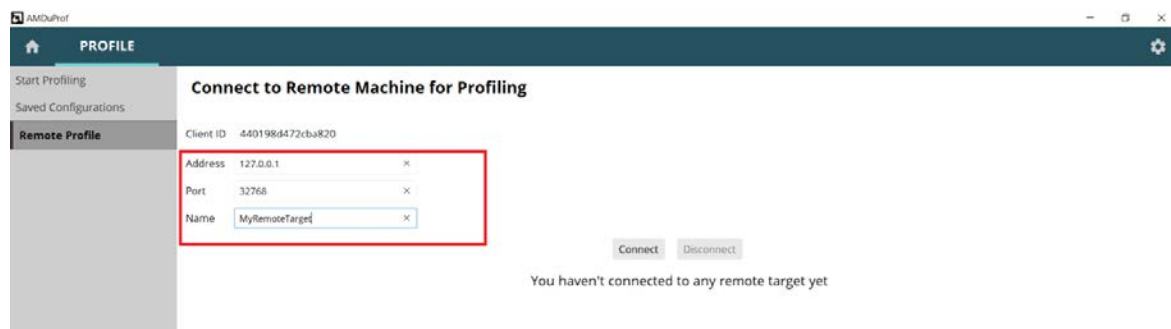


図 80. リモート マシンへの接続

2. **[Connect]** をクリックします。

数秒後に、リモート ターゲット データが表示されます。この後のプロファイリング手順またはインポート手順はすべて、ローカルとまったく同じです。接続が完了すると、次に示すように、指定した IP、ポート、名前が保存されます。

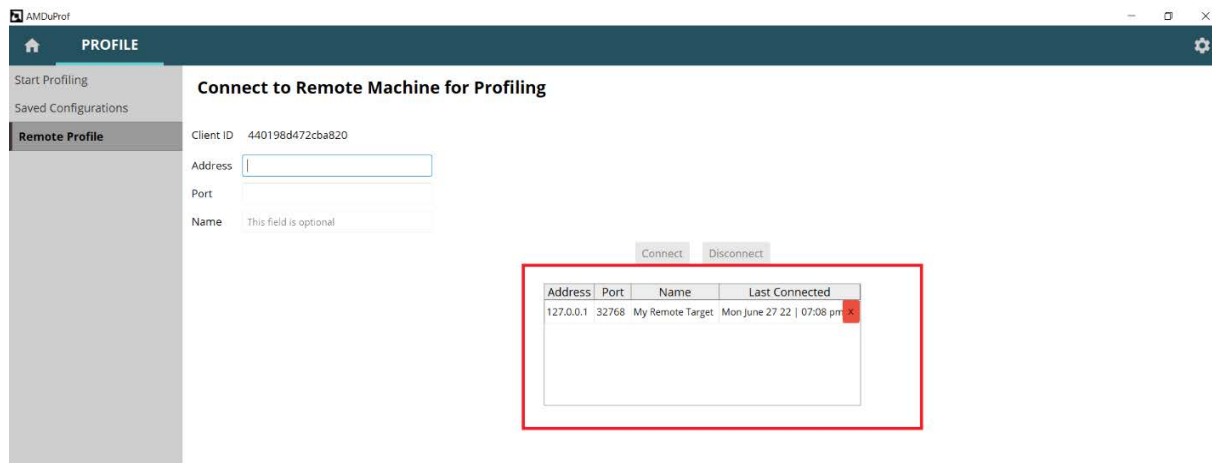


図 81. リモート ターゲットのデータ

IP アドレスを含む任意のテーブル エントリをダブルクリックすると、対応する情報が読み込まれ、目的のリモート ターゲットに接続できます。

接続すると、次のようにタイトル バーにリモート ターゲットへの接続が示され、[Remote Profile] ページで [Connect] ボタンの代わりに [Disconnect] ボタンが有効になります。

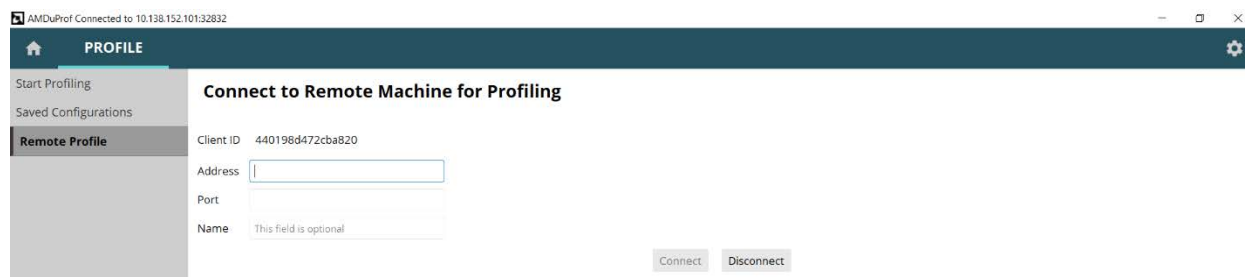


図 82. [Disconnect] ボタン

10.5 制限

- リモート ターゲットに接続した後、GUI ですべての **[Browse]** ボタンは無効のままになります。必要な場合は、URI パスをコピー / ペーストまたは入力します。
- ローカルでのプロファイリング後に GUI を終了しないまま、リモート ターゲットに接続を試みた場合、GUI がクラッシュする場合があります。このため、リモート接続が必要な場合は、ローカル プロファイリング後に GUI を終了することを推奨します。
- ローカル データが必要ではなく、同じリモート ターゲットに頻繁に接続する場合は、次のコマンドを使用すると、リモート ターゲットに直接接続できます (ターゲットが実行中の場合)。

```
AMDuProf <ip_address> <port>
```

例: AMDuProf 127.0.0.1 32768

- クライアント (GUI インスタンス) から AMDProfilerService インスタンスに接続できます。ただし、1 人のユーザーによって GUI インスタンスが複数起動された場合、そのうちの 1 つだけが接続に成功します。クライアント ID が異なるため、複数のユーザーが同じ AMDProfilerService に接続できます。
- AMDProfilerService のインスタンスを複数起動できます。ただし、これらが同じ IP アドレスにバインドされている場合でも、それぞれ異なるポートを使用する必要があります。
- ターゲット システムでファイアウォールが有効になっている場合、リモート プロファイリングの接続確立に失敗する場合があります。このような場合は、ファイアウォールを無効にするか、ターゲット システムのファイアウォール ルールに AMDProfilerService の例外を追加してから、再接続を試行します。もう 1 つの失敗の理由には、ポート番号を使用できないケースがあります。この場合の原因として、ネットワーク設定、ファイアウォール設定、その他のプログラムによる使用可能ポートのブロックが考えられます。
- リモート プロファイリングでは、MPI アプリケーションのプロファイリングはサポートされていません。

第 11 章 AMD uProf の仮想化サポート

11.1 概要

AMD uProf は、仮想化環境のプロファイリングをサポートしています。プロファイリング機能を使用できるかどうかは、ハイパーバイザー マネージャーによって仮想化されるカウンターによって異なります。現在、AMD uProf は、次に示すハイパーバイザーをサポートしています (これらの仮想化環境で Linux および Windows OS をゲストとして使用)。

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

各種ハイパーバイザーでの機能サポート マトリックスを次に示します。

表 62. AMD uProf の仮想化サポート

機能	Microsoft Hyper-V			KVM		VMware ESXi		Citrix Xen	
	ホスト ルート パーティ ション (システム モード)	ホスト ルート パーティ ション	ゲスト VM	ホスト	ゲスト VM	ホスト	ゲスト VM	ホスト	ゲスト VM
CPU プロファイリング									
時間ベース プロファイリング (TBP)	あり	あり	あり	あり	あり	あり	あり	あり	あり
マイクロ アーキテクチャ 解析 (EBP)	あり	あり	あり	あり	あり	あり	あり	なし	なし
命令ベースの サンプリング (IBS)	あり	なし	なし	なし	なし	なし	なし	なし	なし
キャッシュ解析	あり	なし	なし	なし	なし	なし	なし	なし	なし
HPC - MPI コード プロファイリング	あり	あり	あり	あり	あり	あり	あり	あり	あり
HPC - OpenMP トレース	あり	あり	あり	あり	あり	あり	あり	あり	あり
HPC - MPI トレース	あり	あり	あり	あり	あり	あり	あり	あり	あり
OS トレース	あり	あり	あり	あり	あり	あり	あり	あり	あり

表 62. AMD uProf の仮想化サポート (続き)

機能	Microsoft Hyper-V			KVM		VMware ESXi		Citrix Xen	
	ホスト ルート パーティ ション (システム モード)	ホスト ルート パーティ ション	ゲスト VM	ホスト	ゲスト VM	ホスト	ゲスト VM	ホスト	ゲスト VM
消費電力プロファイリング									
ライブ消費電力 プロファイリング	なし	なし	なし	なし	なし	なし	なし	なし	なし
消費電力 アプリケーション 解析	なし	なし	なし	なし	なし	なし	なし	なし	なし
ユーザー インターフェイス									
グラフィカル インターフェイス	あり	あり	あり	あり	あり	あり	あり	あり	あり
コマンド ライン	あり	あり	あり	あり	あり	あり	あり	あり	あり
API									
プロファイル制御 API	あり	あり	あり	あり	あり	あり	あり	あり	あり
消費電力 プロファイラー API	なし	なし	なし	なし	なし	なし	なし	なし	なし
システム解析									
AMDuProfPCM	あり	あり	あり	あり	あり	あり	あり	なし	なし
AMDuProfSys	あり	あり	あり	あり	あり	あり	あり	なし	なし

注記: 各ハイパーバイザーでゲスト VM を設定する間は、仮想化ハードウェア カウンターを有効にしておく必要があります。

11.2 CPU プロファイリング

CPU プロファイリングでは、次がサポートされます。

- ゲスト VM からゲスト VM のプロファイリング。
- ホスト システム (KVM ハイパーバイザー) からゲスト VM のプロファイリング。

11.2.1 ゲスト VM からゲスト VM のプロファイリング

サポートされるすべてのホストおよびゲスト VM で、時間ベース プロファイリングを実行できます。ただし、ハードウェア カウンター プロファイリングは、ハイパーバイザーによって公開される vPMU に完全に左右します。

11.2.2 ホスト システム (KVM ハイパーバイザー) からゲスト VM のプロファイリング

この機能は、ホストから KVM ゲスト OS カーネルおよびカーネル モジュール (*.ko) のプロファイリングをサポートしています。次の機能がサポートされます。

- ゲスト OS での PMU サンプルの収集
- ゲスト OS および/またはホスト OS のプロファイリング
- システム全体のプロファイリングによる、KVM ゲストとその他の実行中プロセスのプロファイリング

次の機能はサポートされていません。

- コール スタック
- プロセスの接続
- アプリケーションの起動

11.2.3 ゲスト カーネル モジュールをプロファイリングするためのホスト システムの準備

ゲスト OS でプロファイリングを開始する前に、次に示すファイルをホスト マシンにコピーして、ゲスト VM のシンボルが解決されるようにする必要があります。

1. `/proc/kallsyms` および `/proc/modules` を、ゲスト OS からホスト マシンにコピーします。
2. ホスト システムのフォルダーで、ゲスト `vmlinux` とカーネル ソースをコピーします。

これらのファイルは、PID が `--guest-kvm` オプションの引数として入力されたゲスト VM に属している必要があります。

11.2.4 AMD uProf CLI のプロファイリング オプション

AMD uProf CLI には、ホスト OS からのゲスト OS プロファイリングをサポートするため、次のオプションが含まれています。

```
$ ./AMDuProfCLI collect [--kvm-guest <pid>] [--guest-kallsyms <path>] [--guest-modules <path>]  
[--guest-search-path <path>] ....
```

次の表に、Collect コマンドの各種オプションを示します。

表 63. AMD uProf CLI Collect コマンドのオプション

引数	オプション	説明
--kvm-guest	プロファイリングする qemu-kvm プロセスの PID	ゲスト側のパフォーマンス プロファイルの収集。 このオプションは、KVM ゲスト シンボル情報を収集します。
--guest-kallsyms	ローカル ホストにコピーされた <i>guest/proc/kallsyms</i> のパス	ゲスト OS の <i>/proc/kallsyms</i> ファイルのコピー。 AMD uProf はこれを読み取って、ゲスト カーネル シンボルを取得します。このファイルはゲスト OS からコピーできます。
--guest-modules	ローカル ホストにコピーされた <i>guest/proc/modules</i> のパス	ゲスト OS の <i>/proc/modules</i> ファイルのコピー。 AMD uProf はこれを読み取って、ゲスト カーネル のモジュール情報を取得します。このファイルはゲスト OS からコピーできます。
--guest-search-path	ローカル ホストにコピーされた ゲスト <i>vmlinux</i> とカーネル ソースのパス	ゲスト OS の <i>vmlinux</i> および検索ディレクトリ。 AMD uProf はこれを読み取って、ゲスト カーネル のモジュール情報を解決します。このファイルはゲスト OS からコピーできます。

11.2.5 例

- kvm ゲスト OS の PID を取得します。

```
$ ps aux | grep kvm
```

- 10 秒間にわたり、(ゲスト kallsyms およびゲスト カーネル モジュールに対して) pmcx76 イベント データを収集します。

```
$ ./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
```

収集したデータからレポートを生成します。

```
$ ./AMDuProfCLI report -i /tmp/cpuprof-76-guest-only/AMDuProf-SWP-EBP_Nov-08-2021_15-00-33
```

- 10 秒間にわたり、(ゲスト kallsyms に対して) pmcx76 イベント データを収集します。

```
$ ./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
```

収集したデータからレポートを生成します。

```
$ ./AMDuProfCLI report -i /tmp/cpuprof-76-guest-only/AMDuProf-SWP-EBP_Nov-08-2021_15-00-33
```

- 10 秒間にわたり、(ゲスト kallsyms およびゲスト カーネル モジュールに対する) pmcx76 イベント データのサンプルをシステム全体で収集します。

```
$ ./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
```

収集したデータからレポートを生成します。

```
$ ./AMDuProfCLI report -i /tmp/cpuprof-76-guest-only/AMDuProf-SWP-EBP_Nov-08-2021_15-00-33
```

- 10 秒間にわたり、(ゲスト kallsyms に対する) pmcx76 イベント データのサンプルをシステム全体で収集します。

```
$ ./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
```

収集したデータからレポートを生成します。

```
$ ./AMDuProfCLI report -i /tmp/cpuprof-76-guest-only/AMDuProf-SWP-EBP_Nov-08-2021_15-00-33
```

11.3 AMDuProfPcm

AMDuProfPcm は、ホストまたはゲスト OS によって提供される次のハードウェアおよび OS プリミティブに基づいています。この情報を取得するには、/AMDuProfCLI info --system コマンドを実行して、次のセクションを確認します。

```
[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
```

Linux 環境では、msr モジュールが使用できることを確認します。このモジュールは、次のコマンドを使用してロードできます。

```
$ modprobe msr
```

11.4 AMDuProfSys

AMDuProfSys は、ホストまたはゲスト OS によって提供される次のハードウェアおよび OS プリミティブに基づいています。この情報を取得するには、/AMDuProfCLI info --system コマンドを実行して、次のセクションを確認します。

```
[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
```

Linux 環境では、Linux カーネルの perf モジュールとユーザー スペース ツールが使用できることを確認します。

第 12 章 プロファイル制御 API

12.1 AMDProfileControl API

AMDProfileControl API により、プロファイリング スコープを、ターゲット アプリケーション内の特定のコード部分に限定できるようになります。

AMDProfileControl API をアプリケーション解析のために使用できるのは、AMDuProfCLI および GUI と共に使用する場合のみです。次のツールと一緒に使用できません。

- 消費電力プロファイラー
- システム解析ツール (uProfPcm および uProfSys)

アプリケーションのプロファイリング中は通常、アプリケーション実行の制御フロー全体に対して、つまり、アプリケーション実行の開始から終了までサンプルが収集されます。制御 API を使用すると、プロファイラーが、CPU 処理の多いループやホット関数など、アプリケーションの特定部分のみに対してデータを収集するようにできます。

必要なコード領域のみに対するプロファイリングの有効化/無効化をターゲット アプリケーションにインストールメント化した後、そのアプリケーションを再コンパイルする必要があります。

ヘッダー ファイル

必要な API を宣言したヘッダー ファイル *AMDProfileController.h* をアプリケーションにインクルードする必要があります。このファイルは、AMD uProf インストールパスの下の *include* ディレクトリにあります。

スタティック ライブラリ

インストールメント化したアプリケーションには、AMDProfileController スタティック ライブラリをリンクする必要があります。このライブラリの場所を次に示します。

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 プロファイル制御 API

CPU プロファイル制御 API により、C または C++ アプリケーションで、CPU プロファイル データの収集を一時停止して再開できます。

amdProfileResume

インストールメント化されたターゲット アプリケーションが AMDuProf/AMDuProfCLI から起動される際、プロファイリングは一時停止ステートになり、アプリケーションがこの再開 API を呼び出すまで、プロファイル データは収集されません。

```
bool amdProfileResume ();
```

amdProfilePause

インストルメント化されたターゲット アプリケーションでのプロファイル データ収集を一時停止する必要があるときは、この API を呼び出します。

```
bool amdProfilePause ();
```

これらの API はアプリケーション内で複数回呼び出すことができます。再開 - 一時停止呼び出しのネスト化はサポートされていません。AMD uProf は、それぞれの再開 - 一時停止 API ペア内でコードをプロファイリングします。これらの API を追加したら、プロファイル セッションを開始する前にターゲット アプリケーションをコンパイルする必要があります。

12.1.2 API の使用

ヘッダー ファイル *AMDProfileController.h* をインクルードし、再開および一時停止 API をコード内で呼び出します。再開 - 一時停止 API ペアの間にカプセル化されたコードは、CPU プロファイラーによってプロファイリングされます。

これらの API の特徴は次のとおりです。

- コードの異なる箇所をプロファイリングするために、複数回呼び出すことができます。
- 複数の関数にまたがることができます。つまり、ある関数から再開を呼び出して、別の関数から停止を呼び出すことができます。
- スレッド間にまたがることができます。つまり、あるスレッドから再開を呼び出して、同じターゲット アプリケーションの別のスレッドから停止を呼び出すことができます。

次のコード スニペットでは、CPU プロファイリング データの収集が `multiply_matrices()` 関数の実行に限定されています。

```
#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 インストルメント化されたターゲット アプリケーションのコンパイル

Windows

Microsoft Visual Studio でアプリケーションをコンパイルするには、構成プロパティを更新してヘッダー ファイルのパスを含め、*AMDProfileController.lib* ライブラリとリンクします。

Linux

Linux で g++ を使用して C++ アプリケーションをコンパイルするには、次のコマンドを使用します。

```
$ g++ -std=c++11 -g <sourcefile.cpp> -I <AMDuProf-install-dir>/include -L<AMDuProf-install-dir>/lib/x64/ -lAMDProfileController -lrt -pthread
```

注記: g++ でコンパイルする際、-static オプションは使用しないでください。

Linux で gcc を使用して C アプリケーションをコンパイルするには、次のコマンドを使用します。

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

12.1.4 インストルメント化されたターゲット アプリケーションのプロファイリング

AMD uProf GUI

ターゲット アプリケーションをコンパイルしたら、AMD uProf でプロファイル設定を作成し、必要な CPU プロファイル セッション オプションを設定します。CPU プロファイル セッション オプションを設定する際、**[Profile Scheduling]** セクションで **[Are you using Profile Instrumentation API?]** を選択します。

すべての設定を完了したら、CPU プロファイリングを開始します。一時停止ステートでプロファイリングが開始して、ターゲット アプリケーションの実行が始まります。ターゲット アプリケーションから再開 API が呼び出されると、CPU プロファイリングが始まり、ターゲット アプリケーションから一時停止 API が呼び出されるか、アプリケーションが終了されるまで続きます。ターゲット アプリケーションで一時停止 API が呼び出されると、プロファイラーはプロファイリングを停止し、次の制御 API 呼び出しまで待機します。

AMDuProfCLI

CLI からプロファイリングする場合、一時停止ステートでプロファイラーを開始するために、--start-paused オプションを使用する必要があります。

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 制限

CPU プロファイル制御 API は、MPI アプリケーションではサポートされません。

第 13 章 参考資料

13.1 プロファイリングのためのアプリケーションの準備

AMD uProf は、コンパイラによって生成されたデバッグ情報を使用して、さまざまな解析ビューに正しい関数名を表示し、収集されたサンプルをソース ページのソース ステートメントに関連付けします。このように処理しないと、CPU プロファイラーの結果にはアセンブリ コードのみが表示され、わかりにくくなります。

13.1.1 Windows でのデバッグ情報の生成

Microsoft Visual C++ を使用してアプリケーションをリリース モードでコンパイルする場合は、デバッグ情報が生成されてプログラム データベース ファイル (.pdb 拡張子付き) に保存されるようにするため、アプリケーションのコンパイル前に次のオプションを設定します。リリース モードの x64 アプリケーションでデバッグ情報を生成するようにコンパイラ オプションを設定するには、次の手順に従います。

1. プロジェクトを右クリックし、メニューから **[Properties]** を選択します。
2. **[Configuration]** ドロップダウンで、**[Active(Release)]** を選択します。
3. **[Platform]** ドロップダウンで、**[Active(Win32)]** または **[Active(x64)]** を選択します。
4. 左側のプロジェクト ペインで **[Configuration Properties]** を開きます。
5. **[C/C++]** を開いて **[General]** を選択します。
6. 作業ペインで、**[Debug Information Format]** を選択します。

7. ドロップダウンから **[Program Database (/ZI)]** または **[Program Database for Edit & Continue (/ZI)]** を選択します。

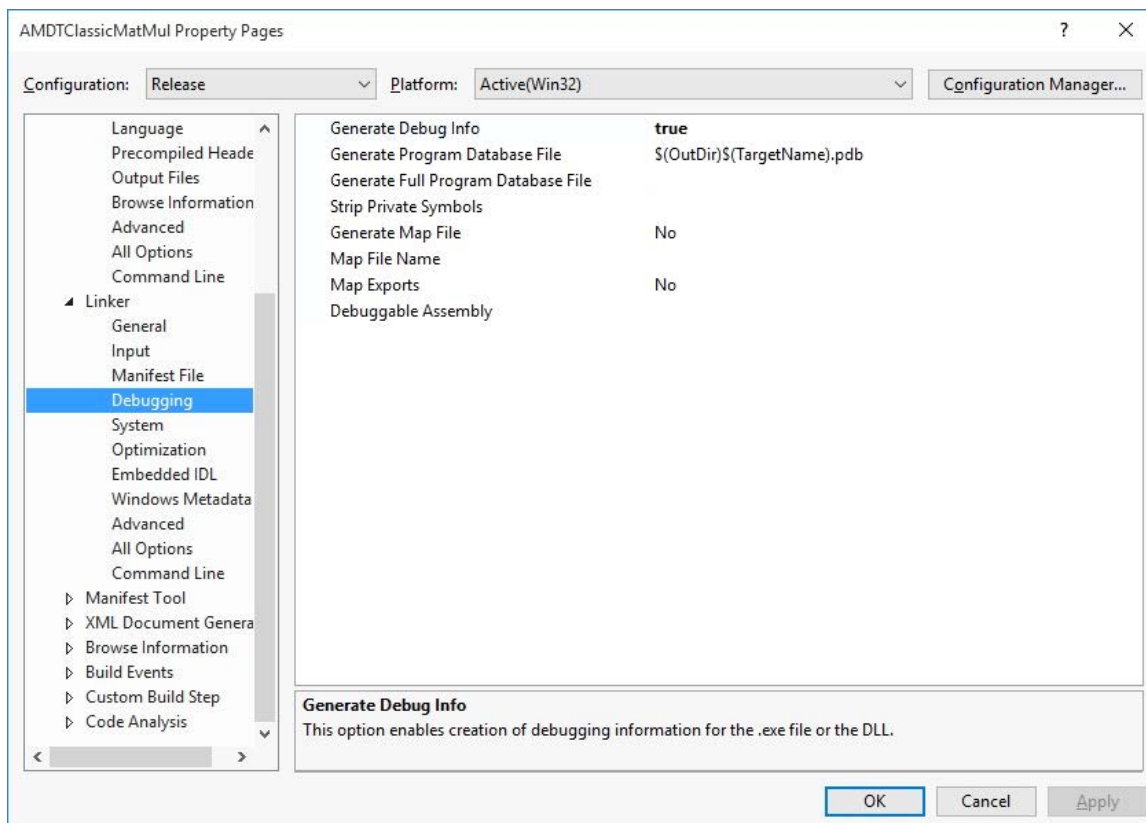


図 83. [AMDTClassicMatMul Property Page]

8. プロジェクト ペインで **[Linker]** を開き、**[Debugging]** を選択します。
9. **[Generate Debug Info]** ドロップダウンで、**[/DEBUG]** を選択します。

13.1.2 Linux でのデバッグ情報の生成

コンパイラがデバッグ情報を生成できるようにするには、アプリケーションのコンパイル時に **-g** オプションを指定する必要があります。makefile または各ビルド スクリプトを適宜変更します。

13.2 CPU プロファイリング

AMD uProf の CPU パフォーマンス プロファイリングは、サンプリング ベースのアプローチに従って、プロファイルデータを定期的に収集します。その際、AMD x86 ベース プロセッサ ファミリで提供されているさまざまなソフトウェアおよびハードウェア リソースを使用します。CPU プロファイリングで使用されるのは、OS タイマー、HW パフォーマンス モニター カウンター (PMC)、HW IBS 機能です。

次のセクションでは、CPU プロファイリングに関連する重要な各種のコンセプトについて説明します。

13.2.1 ハードウェア ソース

パフォーマンス モニター カウンター (PMC)

AMD プロセッサには、パフォーマンス モニター カウンター (PMC) が備わっており、CPU コア内での各種マイクロアーキテクチャ イベントを監視するのに使用できます。PMC カウンターは次の 2 つのモードで使用されます。

- カウンティング モード: このモードのカウンターは、CPU コア内で発生する特定のイベントをカウントするために使用されます。
- サンプリング モード: このモードのカウンターは、特定回数のイベントをカウントするようにプログラムされています。カウントが適切な回数 (サンプリング間隔) に達すると、割り込みがトリガーされます。割り込み処理の間は、CPU プロファイラーがプロファイル データを収集します。

各プロセッサで利用できるハードウェア パフォーマンス イベント カウンターの数は、インプリメンテーションによって異なります。ハードウェア パフォーマンス カウンターの正確な数については、該当プロセッサのプロセッサ プログラミング リファレンス (PPR - <https://developer.amd.com/resources/developer-guides-manuals/>) を参照してください。オペレーティング システムおよび/または BIOS では、1 つ以上のカウンターを内部使用のために予約できます。このため、実際に使用可能なハードウェア カウンター数は、ハードウェア カウンター自体の数よりも少ない場合があります。CPU プロファイラーは、使用可能なすべてのカウンターをプロファイリングに使用します。

命令ベースのサンプリング (IBS)

IBS はコード プロファイリング メカニズムの 1 つであり、プロセッサが、プログラムされた時間間隔の経過後に、ランダムな命令フェッチまたはマイクロ オペレーションを選択して、そのオペレーションに関する具体的なパフォーマンス情報を記録できるようにするものです。オペレーションが完了すると、IBS 制御 MSR によって指定されたとおりに割り込みが生成されます。続いて、割り込みハンドラーが、そのオペレーションに対して記録されたパフォーマンス情報を読み出します。

IBS メカニズムは次の 2 つの部分に分けられます。

- 命令フェッチ パフォーマンス
- 命令実行パフォーマンス

命令フェッチ サンプリングは、フェッチされた命令の命令 TLB と命令キャッシュ動作に関する情報を提供します。

命令実行サンプリングは、マイクロ オペレーション実行の動作に関する情報を提供します。

命令フェッチ パフォーマンス向けに収集されるデータは、命令実行パフォーマンス向けに収集されるデータとは無関係です。

命令実行パフォーマンスは、1 つの命令に関連付けられた 1 つのマイクロ オペレーションをタグ付けすることでプロファイリングされます。複数のマイクロ オペレーションにデコードされる命令の場合、命令に関連付けられたマイクロ オペレーションのうち、どれにタグ付けされているかによって、異なるパフォーマンスデータが返されます。これらのマイクロ オペレーションは、次の命令の RIP に関連付けられています。

このモードで、CPU プロファイラーは、AMD プロセッサがサポートする IBS HW を使用して、命令によるプロセッサとメモリ サブシステムへの影響を観察します。IBS では、ハードウェア イベントは、イベントを発生させた命令にリンクされています。また、データ キャッシュ レイテンシなどの各種メトリクスを算出するために、CPU プロファイラーがハードウェア イベントを使用します。

L3 キャッシュ パフォーマンス モニター カウンター (L3PMC)

コア コンプレックス (CCX) は、L3 キャッシュ リソースを共有する CPU コアで構成されるグループです。1 つの CCX に含まれるすべてのコアが、単一の L3 キャッシュを共有します。L3PMC は、AMD “Zen” ベースのプロセッサで、L3 リソースのパフォーマンスを監視するために使用できます。詳細は、該当するプロセッサの PPR を参照してください。

データ ファブリック パフォーマンス モニター カウンター (DFPMC)

AMD “Zen” ベースのプロセッサでは、データ ファブリック リソースのパフォーマンスを監視するために DFPMC を使用できます。詳細は、該当するプロセッサのプロセッサ プログラミング リファレンス (PPR) を参照してください。

13.2.2 プロファイリングの概念

サンプリング

サンプリング プロファイラーの動作のベースとなるロジックは、多くの時間を消費する (または、サンプリング イベントを最も多く引き起こす) プログラム部分はサンプル数が多くなるということです。なぜなら、これらは、CPU プロファイラーによるサンプル取得中に実行される確率が高いからです。

サンプリング間隔

2 つのサンプルを収集する合間の時間をサンプリング間隔と呼びます。TBP の場合、時間間隔が 1 ミリ秒であれば、各プロセッサ コアで、毎秒およそ 1,000 TBP サンプルが収集されます。

次に示すように、サンプリング間隔の目的は、サンプリング イベントとして使用されるリソースによって異なります。

- OS タイマー — サンプリング間隔は、ミリ秒単位の時間です。
- PMC イベント — サンプリング間隔は、サンプリング イベントの発生回数です。
- IBS — 処理される命令の数で、この数に達するとタグ付けされます。

サンプリング間隔が小さいと、収集されるサンプル数とデータ収集時のオーバーヘッドが増加します。プロファイル データはワークロードが実行されるのと同じシステムで収集されるため、サンプリング頻度が上がるとプロファイリングの侵襲性も高くなります。また、サンプリング間隔を非常に小さくすると、システムが不安定になる場合があります。

サンプリング ポイント: 特定のサンプリング イベントのサンプリング間隔が終了し、サンプリング ポイントに達すると、命令ポインター、プロセス ID、スレッド ID、コールスタックなどの各種プロファイル データが、割り込みハンドラーによって収集されます。

イベント カウンターの多重化

監視対象 PMC イベントの数が、使用可能なパフォーマンス カウンター数以下である場合、各イベントを 1 つのカウンターに割り当てて、常時監視できます。単一プロファイル測定で、監視対象イベントの数が使用可能なカウンター数を超える場合、CPU プロファイラーは、使用可能な HW PMC カウンターをタイムシェアリングします。これをイベント カウンターの多重化と呼びます。この場合、監視されるイベント数が多くなり、イベントごとの実際のサンプル数が減るため、データ精度は低下します。CPU プロファイラーはサンプル カウントを自動スケーリングすることで、このイベント カウンターの多重化を補足します。たとえば、イベントが 50% の時間監視されている場合、CPU プロファイラーはイベント サンプル数を 2 倍にスケーリングします。

13.2.3 プロファイル タイプ

プロファイル タイプは、プロファイル データを収集するのに使用される、ハードウェアまたはソフトウェアのサンプリング イベントに基づいて分類されます。

時間ベース プロファイル (TBP)

このプロファイリングでは、指定された OS タイマー間隔に基づいて、定期的にプロファイル データが収集されます。プロファイル対象アプリケーションのホットスポットを特定するために使用されます。

イベント ベース プロファイル (EBP)

このプロファイリングで、CPU プロファイラーは PMC を使用して、AMD の x86 ベース プロセッサでサポートされる各種のマイクロ アーキテクチャ イベントを監視します。プロファイル対象アプリケーションに含まれる CPU およびメモリ関連のパフォーマンスの問題を特定するのに役立ちます。CPU プロファイラーは、事前定義された EBP プロファイル設定をいくつか提供しています。プロファイル対象アプリケーション (またはシステム) のある側面を解析するために、特定の関連イベントがグループ化されており、まとめて監視されます。CPU プロファイラーは、Assess Performance や Investigate Branching など、一連の事前定義イベント設定を提供しています。これらの事前定義設定から任意の設定を選択して、アプリケーションの実行時特性をプロファイリングして解析できます。また、プロファイル対象イベントのカスタム設定を独自に作成することもできます。

このプロファイル モードでは、サンプリング割り込みが発生する時点と、サンプリング対象の命令アドレスが収集される時点の間で、スキッドと呼ばれる遅延が発生します。このスキッドにより、サンプリング割り込みをトリガーした実際の命令付近にサンプルが分散します。そのため、サンプル分布が不正確になり、イベントの原因として間違った命令が特定されることがよくあります。

命令ベースのサンプリング (IBS)

このプロファイリングで、CPU プロファイラーは、AMD の x86 ベース プロセッサがサポートする IBS HW を使用して、命令によるプロセッサおよびメモリ サブシステムへの影響を観察します。IBS では、HW イベントは、イベントが発生させた命令にリンクされています。また、データ キャッシュ レイテンシなどの各種メトリクスを算出するために、CPU プロファイラーは HW イベントを使用します。

カスタム プロファイル

カスタム プロファイルにより、HW PMC イベント、OS タイマー、IBS サンプリング イベントを組み合わせられるようになります。

13.2.4 事前定義コア PMC イベント

次の表に、AMD “Zen” プロセッサのコア パフォーマンス イベントの一部を示します。

表 64. 事前定義コア PMC イベント

イベント ID、 ユニット マスク	イベント 略称	名前と説明
AMD 第 2 世代 EPYC™ プロセッサ		
0x76、0x00	CYCLES_NOT_IN_HALT	CPU clock cycles not halted スレッドが停止ステートではないときの、CPU サイクルの数。
0xC0、0x00	RETIRED_INST	Retired Instructions 実行からリタイアした命令の数。このカウントには例外と割り込みが含まれます。それぞれの例外または割り込みが 1 つの命令としてカウントされます。
0xC1、0x00	RETIRED_MICRO_OPS	Retired Macro Operations リタイアしたマイクロ オペレーションの数。このカウントには、命令、例外、割り込み、マイクロコード アシストなど、すべてのプロセッサ アクティビティが含まれます。
0xC2、0x00	RETIRED_BR_INST	Retired Branch Instructions リタイアした分岐命令の数。これには、例外と割り込みなど、あらゆる種類のアーキテクチャ制御フロー変更が含まれます。
0xC3、0x00	RETIRED_BR_INST_MISP	Retired Branch Instructions Mispredicted 予測ミスしたリタイア済み分岐命令の数。 注記: EX の直接予測ミスとターゲットの間接予測ミスのみがカウントされます。
0x03、0x08	RETIRED_SSE_AVX_FLOPS	Retired SSE/AVX Flops リタイアした SSE/AVX FLOPS の数。サイクルあたりで記録されるイベント数は、0 ~ 64 までの範囲で異なります。サイクルあたりで 15 を超えるイベントがカウントされる場合があるため、サイクル イベントあたりの増分は大きくなります。単精度と倍精度の両方の FP イベントがカウントされます。
0x29、0x07	L1_DC_ACCESSES_ALL	All Data cache accesses LS ユニットにディスパッチされたロードおよびストア オペレーションの数。これには、メモリ ロードを実行する単一オペレーションのディスパッチ、メモリ ストアを実行する単一オペレーションのディスパッチ、同じメモリ アドレスに対するロードとストアを実行する単一オペレーションのディスパッチがカウントされます。

表 64. 事前定義コア PMC イベント (続き)

イベント ID、 ユニットマスク	イベント略称	名前と説明
0x60、0x10	L2_CACHE_ACCESS_FROM_L1_IC_MISS	L2 cache access from L1 IC miss L1 命令キャッシュ ミスによる L2 キャッシュ アクセス要求。
0x60、0xC8	L2_CACHE_ACCESS_FROM_L1_DC_MISS	L2 cache access from L1 DC miss L1 データ キャッシュ ミスによる L2 キャッシュ アクセス要求。これには、ハードウェアおよびソフトウェア プリフェッチもカウントされます。
0x64、0x01	L2_CACHE_MISS_FROM_L1_IC_MISS	L2 cache miss from L1 IC miss L2 キャッシュをミスするすべての命令キャッシュ フィル要求をカウントします。
0x64、0x08	L2_CACHE_MISS_FROM_L1_DC_MISS	L2 cache miss from L1 DC miss L2 キャッシュをミスするすべてのデータ キャッシュ フィル要求をカウントします。
0x71、0x1F	L2_HWPF_HIT_IN_L3	L2 Prefetcher Hits in L3 L2 パイプラインに受け入れられたすべての L2 プリフェッチで、L2 キャッシュをミスし、L3 をヒットするものをカウントします。
0x72、0x1F	L2_HWPF_MISS_IN_L2_L3	L2 Prefetcher Misses in L3 L2 パイプラインに受け入れられたすべての L2 プリフェッチで、L2 および L3 キャッシュをミスするものをカウントします。
0x64、0x06	L2_CACHE_HIT_FROM_L1_IC_MISS	L2 cache hit from L1 IC miss L2 キャッシュをヒットするすべての命令キャッシュ フィル要求をカウントします。
0x64、0x70	L2_CACHE_HIT_FROM_L1_DC_MISS	L2 cache hit from L1 DC miss L2 キャッシュをヒットするすべてのデータ キャッシュ フィル要求をカウントします。
0x70、0x1F	L2_HWPF_HIT_IN_L2	L2 cache hit from L2 HW Prefetch L2 パイプラインに受け入れられたすべての L2 プリフェッチで、L2 キャッシュでヒットするものをカウントします。
0x43、0x01	L1_DEMAND_DC_REFILLS_LOCAL_L2	L1 demand DC fills from L2 ローカル L2 キャッシュからコアへのデマンド データ キャッシュ (DC) フィル。
0x43、0x02	L1_DEMAND_DC_REFILLS_LOCAL_CACHE	L1 demand DC fills from local CCX 同じ CCX の同じキャッシュまたは同じパッケージ (ノード) に含まれる異なる CCX のキャッシュからのデマンド データ キャッシュ (DC) フィル。

表 64. 事前定義コア PMC イベント (続き)

イベント ID、 ユニットマスク	イベント略称	名前と説明
0x43、0x08	L1_DEMAND_DC_REFILLS_LOCAL_DRAM	L1 demand DC fills from local Memory 同じパッケージ (ノード) 内で接続された DRAM または IO からのデマンド データ キャッシュ (DC) フィル。
0x43、0x10	L1_DEMAND_DC_REFILLS_REMOTE_CACHE	L1 demand DC fills from remote cache 異なるパッケージ (ノード) に含まれる CCX のキャッシュからのデマンド データ キャッシュ (DC) フィル。
0x43、0x40	L1_DEMAND_DC_REFILLS_REMOTE_DRAM	L1 demand DC fills from remote Memory 異なるパッケージ (ノード) 内で接続された DRAM または IO からのデマンド データ キャッシュ (DC) フィル。
0x43、0x5B	L1_DEMAND_DC_REFILLS_ALL	L1 demand DC refills from all data sources すべてのデータ ソースからのデマンド データ キャッシュ (DC) フィル。
0x60、0xFF	L2_REQUESTS_ALL	All L2 cache requests
0x84、0x00	L1_ITLB_MISSES_L2_HITS	L1 TLB miss L2 TLB hit L1 命令変換ルックアサイド バッファ (ITLB) でミスするが、L2 ITLB でヒットする命令フェッチ。
0x85、0x07	L2_ITLB_MISSES	L1 TLB miss L2 TLB miss ページ テーブル ウォークによる ITLB リロード。 テーブル ウォーク要求の対象は、L1-ITLB ミスと L2-ITLB ミスです。
0x45、0xFF	L1_DTLB_MISSES	L1 DTLB miss ロード/ストアのマイクロオペレーションによりミスする L1 データ変換ルックアサイド バッファ (DTLB)。このイベントでは、L2-DTLB ヒットと L2-DTLB ミスの両方がカウントされます。
0x45、0xF0	L2_DTLB_MISSES	L1 DTLB miss ロード/ストアのマイクロオペレーションによりミスした L2 データ変換ルックアサイド バッファ (DTLB)。
0x47、0x00	MISALIGNED_LOADS	Misaligned Loads ミスアライン ロードの数。 注記: AMD “Zen 3” コア プロセッサで、このイベントは 64B (キャッシュ ライン横断) および 4K (ページ横断) のミスアライン ロードをカウントします。

表 64. 事前定義コア PMC イベント (続き)

イベント ID、 ユニットマスク	イベント略称	名前と説明
0x52、0x03	INEFFECTIVE_SW_PF	Ineffective Software Prefetches プロセッサ コアの外部でデータをフェッチしなかったソフトウェアプリフェッチの数。このイベントは、割り当て済みの要求バッファーマスで一致を見つけたソフトウェア PREFETCH 命令をカウントします。また、DC ヒットを見つけたソフトウェア PREFETCH 命令もカウントします。
AMD EPYC™ 第 3 世代プロセッサ		
0x76、0x00	CYCLES_NOT_IN_HALT	CPU clock cycles not halted スレッドが停止ステートではないときの、CPU サイクルの数。
0xC0、0x00	RETIRED_INST	Retired Instructions 実行からリタイアした命令の数。このカウントには例外と割り込みが含まれます。それぞれの例外または割り込みが 1 つの命令としてカウントされます。
0xC1、0x00	RETIRED_MACRO_OPS	Retired Macro Operations リタイアしたマイクロオペレーションの数。このカウントには、命令、例外、割り込み、マイクロコードアシストなど、すべてのプロセッサアクティビティが含まれます。
0xC2、0x00	RETIRED_BR_INST	Retired Branch Instructions リタイアした分岐命令の数。これには、例外と割り込みなど、あらゆる種類のアーキテクチャ制御フロー変更が含まれます。
0xC3、0x00	RETIRED_BR_INST_MISP	Retired Branch Instructions Mis-predicted 予測ミスしたリタイア済み分岐命令の数。EX の直接予測ミスとターゲットの間接予測ミスのみがカウントされることに注意してください。
0x03、0x08	RETIRED_SSE_AVX_FLOPS	Retired SSE/AVX Flops リタイアした SSE/AVX FLOPS の数。サイクルあたりで記録されるイベント数は、0 ~ 64 までの範囲で異なります。サイクルあたりで 15 を超えるイベントがカウントされる場合があるため、サイクルイベントあたりの増分は大きくなります。単精度と倍精度の両方の FP イベントがカウントされます。

表 64. 事前定義コア PMC イベント (続き)

イベント ID、 ユニットマスク	イベント略称	名前と説明
0x29、0x07	L1_DC_ACCESSES_ALL	All Data cache accesses LS ユニットにディスパッチされたロードおよびストア オペレーションの数。これには、メモリ ロードを実行する単一オペレーションのディスパッチ、メモリ ストアを実行する単一オペレーションのディスパッチ、同じメモリ アドレスに対するロードとストアを実行する単一オペレーションのディスパッチがカウントされます。
0x60、0x10	L2_CACHE_ACCESS_FROM_L1_IC_MISS	L2 cache access from L1 IC miss L1 命令キャッシュ ミスによる L2 キャッシュ アクセス要求。
0x60、0xE8	L2_CACHE_ACCESS_FROM_L1_DC_MISS	L2 cache access from L1 DC miss L1 データ キャッシュ ミスによる L2 キャッシュ アクセス要求。これには、ハードウェアおよびソフトウェア プリフェッチもカウントされます。
0x64、0x01	L2_CACHE_MISS_FROM_L1_IC_MISS	L2 cache miss from L1 IC miss L2 キャッシュでミスするすべての命令キャッシュ フィル要求をカウントします。
0x64、0x08	L2_CACHE_MISS_FROM_L1_DC_MISS	L2 cache miss from L1 DC miss L2 キャッシュでミスするすべてのデータ キャッシュ フィル要求をカウントします。
0x71、0xFF	L2_HWPF_HIT_IN_L3	L2 Prefetcher Hits in L3 L2 パイプラインに受け入れられたすべての L2 プリフェッチで、L2 キャッシュをミスし、L3 をヒットするものをカウントします。
0x72、0xFF	L2_HWPF_MISS_IN_L2_L3	L2 Prefetcher Misses in L3 L2 パイプラインに受け入れられたすべての L2 プリフェッチで、L2 および L3 キャッシュをミスするものをカウントします。
0x64、0x06	L2_CACHE_HIT_FROM_L1_IC_MISS	L2 cache hit from L1 IC miss L2 キャッシュをヒットするすべての命令キャッシュ フィル要求をカウントします。
0x64、0xF0	L2_CACHE_HIT_FROM_L1_DC_MISS	L2 cache hit from L1 DC miss L2 キャッシュをヒットするすべてのデータ キャッシュ フィル要求をカウントします。

表 64. 事前定義コア PMC イベント (続き)

イベント ID、 ユニットマスク	イベント略称	名前と説明
0x70、0xFF	L2_HWPF_HIT_IN_L2	L2 cache hit from L2 HW Prefetch L2 パイプラインに受け入れられたすべての L2 プリフェッチで、L2 キャッシュでヒットするものをカウントします。
0x43、0x01	L1_DEMAND_DC_REFILLS_LOCAL_L2	L1 demand DC fills from L2 ローカル L2 キャッシュからコアへのデマンド データ キャッシュ (DC) フィル。
0x43、0x02	L1_DEMAND_DC_REFILLS_LOCAL_CACHE	L1 demand DC fills from local CCX 同じ CCX に含まれる L3 キャッシュまたは L2 からのデマンド データ キャッシュ (DC) フィル。
0x43、0x04	L1_DC_REFILLS_EXTERNAL_CACHE_LOCAL	L1 DC fills from local external CCX caches 同じパッケージ (ノード) 内の異なる CCX のキャッシュからのデータ キャッシュ (DC) フィル。
0x43、0x08	L1_DEMAND_DC_REFILLS_LOCAL_DRAM	L1 demand DC fills from local Memory 同じパッケージ (ノード) 内で接続された DRAM または IO からのデマンド データ キャッシュ (DC) フィル。
0x43、0x10	L1_DEMAND_DC_REFILLS_EXTERNAL_CACHE_REMOTE	L1 demand DC fills from remote external cache 異なるパッケージ (ノード) に含まれる CCX のキャッシュからのデマンド データ キャッシュ (DC) フィル。
0x43、0x40	L1_DEMAND_DC_REFILLS_REMOTE_DRAM	L1 demand DC fills from remote Memory 異なるパッケージ (ノード) 内で接続された DRAM または IO からのデマンド データ キャッシュ (DC) フィル。
0x43、0x14	L1_DEMAND_DC_REFILLS_EXTERNAL_CACHE	L1 demand DC fills from external caches 同じまたは異なるパッケージ (ノード) 内の異なる CCX のキャッシュからのデマンド データ キャッシュ (DC) フィル要求。
0x43、0x5F	L1_DEMAND_DC_REFILLS_ALL	L1 demand DC refills from all data sources すべてのデータ ソースからのデマンド データ キャッシュ (DC) フィル。
0x44、0x01	L1_DC_REFILLS.LOCAL_L2	L1 DC fills from local L2 ローカル L2 キャッシュからコアへのデータ キャッシュ (DC) フィル。

表 64. 事前定義コア PMC イベント (続き)

イベント ID、 ユニットマスク	イベント略称	名前と説明
0x44、0x02	L1_DC_REFILLS_LOCAL_CACHE	L1 DC fills from local CCX cache 同じ CCX 内の異なる L2 キャッシュまたは同じ CCX に属する L3 キャッシュからのデータ キャッシュ (DC) フィル。
0x44、0x08	L1_DC_REFILLS_LOCAL_DRAM	L1 DC fills from local Memory 同じパッケージ (ノード) 内で接続された DRAM または IO からのデータ キャッシュ (DC) フィル。
0x44、0x04	L1_DC_REFILLS_EXTERNAL_CACHE_LOCAL	L1 DC fills from local external CCX caches 同じパッケージ (ノード) 内の異なる CCX のキャッシュからのデータ キャッシュ (DC) フィル。
0x44、0x10	L1_DC_REFILLS_EXTERNAL_CACHE_REMOTE	L1 DC fills from remote external CCX caches 異なるパッケージ (ノード) に含まれる CCX のキャッシュからのデータ キャッシュ (DC) フィル。
0x44、0x40	L1_DC_REFILLS_REMOTE_DRAM	L1 DC fills from remote Memory 異なるパッケージ (ノード) 内で接続された DRAM または IO からのデータ キャッシュ (DC) フィル。
0x44、0x14	L1_DC_REFILLS_EXTENAL_CACHE	L1 DC fills from local external CCX caches 同じまたは異なるパッケージ (ノード) 内の異なる CCX のキャッシュからのデータ キャッシュ (DC) フィル。
0x44、0x48	L1_DC_REFILLS_DRAM	L1 DC fills from local Memory 同じまたは異なるパッケージ (ノード) 内で接続された DRAM または IO からのデータ キャッシュ (DC) フィル。
0x44、0x50	L1_DC_REFILLS_REMOTE_NODE	L1 DC fills from remote node 異なるパッケージ (ノード) 内の CCX のキャッシュ、または異なるパッケージ (ノード) 内で接続された DRAM/IO からのデータ キャッシュ (DC) フィル。
0x44、0x03	L1_DC_REFILLS_LOCAL_CACHE_L2_L3	L1 DC fills from same CCX ローカル L2 キャッシュから、同じ CCX 内のコアあるいは異なる L2 キャッシュ、または同じ CCX に属する L3 キャッシュへのデータ キャッシュ (DC) フィル。
0x44、0x5F	L1_DC_REFILLS_ALL	L1 DC fills from all the data sources すべてのデータ ソースからのデータ キャッシュ フィル。

表 64. 事前定義コア PMC イベント (続き)

イベント ID、 ユニット マスク	イベント 略称	名前と説明
0x60、0xFF	L2_REQUESTS_ALL	All L2 cache requests
0x84、0x00	L1_ITLB_MISSES_L2_HITS	L1 TLB miss L2 TLB hit L1 命令変換ルックアサイド バッファ (ITLB) でミスするが、L2 ITLB でヒットする命令フェッチ。
0x85、0x07	L2_ITLB_MISSES	L1 TLB miss L2 TLB miss ページ テーブル ウォークによる ITLB リロード。 テーブル ウォーク要求の対象は、L1-ITLB ミスと L2-ITLB ミスです。
0x45、0xFF	L1_DTLB_MISSES	L1 DTLB miss ロード/ストアのマイクロオペレーションによりミスする L1 データ変換ルックアサイド バッファ (DTLB)。このイベントでは、L2-DTLB ヒットと L2-DTLB ミスの両方がカウントされます。
0x45、0xF0	L2_DTLB_MISSES	L1 DTLB miss ロード/ストアのマイクロオペレーションによりミスした L2 データ変換ルックアサイド バッファ (DTLB)。
0x78、0xFF	ALL_TLB_FLUSHES	All TLB flushes
0x47、0x03	MISALIGNED_LOADS	ミスアライン ロードの数。 注記: AMD “Zen 3” コア プロセッサで、このイベントは 64B (キャッシュ ライン横断) および 4K (ページ横断) のミスアライン ロードをカウントします。
0x52、0x03	INEFFECTIVE_SW_PF	Ineffective Software Prefetches プロセッサ コアの外部でデータをフェッチしなかったソフトウェアプリフェッチの数。このイベントは、割り当て済みの要求ミス バッファで一致を見つけたソフトウェア PREFETCH 命令をカウントします。また、DC ヒットを見つけたソフトウェア PREFETCH 命令もカウントします。
AMD EPYC™ 第 4 世代プロセッサ		
0x76、0x00	CYCLES_NOT_IN_HALT	CPU clock cycles not halted スレッドが停止ステートではないときの、CPU サイクルの数。
0xC0、0x00	RETIRED_INST	Retired Instructions 実行からリタイアした命令の数。このカウントには例外と割り込みが含まれます。それぞれの例外または割り込みが 1 つの命令としてカウントされます。

表 64. 事前定義コア PMC イベント (続き)

イベント ID、 ユニットマスク	イベント略称	名前と説明
0xC1, 0x00	RETIRED_MACRO_OPS	Retired Macro Operations リタイアしたマイクロ オペレーションの数。この カウントには、命令、例外、割り込み、マイクロ コード アシストなど、すべてのプロセッサ アク ティビティが含まれます。
0xC2, 0x00	RETIRED_BR_INST	Retired Branch Instructions リタイアした分岐命令の数。これには、例外と割 り込みなど、あらゆる種類のアーキテクチャ制御 フロー変更が含まれます。
0xC3, 0x00	RETIRED_BR_INST_MISP	Retired Branch Instructions Mis-predicted 予測ミスしたリタイア済み分岐命令の数。 注記: EX の直接予測ミスとターゲットの間接予測ミスのみがカ ウントされます。
0x03, 0x1F	RETIRED_SSE_AVX_FLOPS	Retired SSE/AVX Flops リタイアした SSE/AVX FLOPS の数。サイクルあた りで記録されるイベント数は、0 ~ 64 までの範囲で 異なります。サイクルあたりで 15 を超えるイベン トがカウントされる場合があるため、サイクル イベ ントあたりの増分は大きくなります。単精度と倍精 度の両方の FP イベントがカウントされます。
0x29, 0x07	L1_DC_ACCESSES_ALL	All Data Cache Accesses LS ユニットにディスパッチされたロードおよびス トア オペレーションの数。次を実行する単一オペ レーションのディスパッチがカウントされます。 • メモリ ロード • メモリ ストア • 同じメモリ アドレスに対するロードおよびストア
0x60, 0x10	L2_CACHE_ACCESS_FROM_L1_IC_MISS	L2 cache access from L1 IC miss L1 命令キャッシュ ミスによる L2 キャッシュ アク セス要求。
0x60, 0xE8	L2_CACHE_ACCESS_FROM_L1_DC_MISS	L2 cache access from L1 DC miss L1 データ キャッシュ ミスによる L2 キャッシュ ア クセス要求。これには、ハードウェアおよびソフ トウェアプリフェッチもカウントされます。
0x64, 0x01	L2_CACHE_MISS_FROM_L1_IC_MISS	L2 cache miss from L1 IC miss L2 キャッシュでのすべての命令キャッシュ フィル 要求ミスがカウントされます。
0x64, 0x08	L2_CACHE_MISS_FROM_L1_DC_MISS	L2 cache miss from L1 DC miss L2 キャッシュでのすべてのデータ キャッシュ フィ ル要求ミスがカウントされます。

表 64. 事前定義コア PMC イベント (続き)

イベント ID、 ユニットマスク	イベント略称	名前と説明
0x71、0xFF	L2_HWPF_HIT_IN_L3	L2 Prefetcher Hits in L3 L2 パイプラインに受け入れられたすべての L2 プリフェッチで、L2 キャッシュをミスし、L3 をヒットするものをカウントします。
0x72、0xFF	L2_HWPF_MISS_IN_L2_L3	L2 Prefetcher Misses in L3 L2 パイプラインに受け入れられたすべての L2 プリフェッチで、L2 および L3 キャッシュをミスするものをカウントします。
0x64、0x06	L2_CACHE_HIT_FROM_L1_IC_MISS	L2 cache hit from L1 IC miss L2 キャッシュをヒットするすべての命令キャッシュ フィル要求をカウントします。
0x64、0xF0	L2_CACHE_HIT_FROM_L1_DC_MISS	L2 cache hit from L1 DC miss L2 キャッシュをヒットするすべてのデータ キャッシュ フィル要求をカウントします。
0x70、0xFF	L2_HWPF_HIT_IN_L2	L2 cache hit from L2 HW Prefetch L2 パイプラインに受け入れられたすべての L2 プリフェッチで、L2 キャッシュをヒットするものをカウントします。
0x43、0x01	L1_DEMAND_DC_REFILLS_LOCAL_L2	L1 demand DC fills from L2 ローカル L2 キャッシュからコアへのデマンド データ キャッシュ (DC) フィル。
0x43、0x02	L1_DEMAND_DC_REFILLS_LOCAL_CACHE	L1 demand DC fills from local CCX 同じ CCX に含まれる L3 キャッシュまたは L2 からのデマンド データ キャッシュ (DC) フィル。
0x43、0x04	L1_DEMAND_DC_REFILLS_EXTERNAL_CACHE_LOCAL	L1 DC fills from local external CCX caches 同じパッケージ (ノード) 内の異なる CCX のキャッシュからの DC フィル。
0x43、0x08	L1_DEMAND_DC_REFILLS_LOCAL_DRAM	L1 demand DC fills from local Memory 同じパッケージ (ノード) 内で接続された DRAM または IO からのデマンド DC フィル。
0x43、0x10	L1_DEMAND_DC_REFILLS_EXTERNAL_CACHE_REMOTE	L1 demand DC fills from remote external cache 異なるパッケージ (ノード) 内の CCX キャッシュからのデマンド DC フィル。
0x43、0x40	L1_DEMAND_DC_REFILLS_REMOTE_DRAM	L1 demand DC fills from remote Memory 異なるパッケージ (ノード) 内で接続された DRAM または IO からのデマンド DC フィル。

表 64. 事前定義コア PMC イベント (続き)

イベント ID、 ユニットマスク	イベント略称	名前と説明
0x43、0x14	L1_DEMAND_DC_REFILLS_EXTERNAL_CACHE	L1 demand DC fills from external caches 同じまたは異なるパッケージ (ノード) 内の異なる CCX のキャッシュからのデマンド DC フィル。
0x43、0xDF	L1_DEMAND_DC_REFILLS_ALL	L1 demand DC refills from all data sources すべてのデータ ソースからのデマンド DC フィル。
0x44、0x01	L1_DC_REFILLS_LOCAL_L2	L1 DC fills from local L2 ローカル L2 キャッシュからコアへの DC フィル。
0x44、0x02	L1_DC_REFILLS_LOCAL_CACHE	L1 DC fills from local CCX cache 同じ CCX 内の異なる L2 キャッシュまたは同じ CCX に属する L3 キャッシュからの DC フィル。
0x44、0x08	L1_DC_REFILLS_EXTERNAL_CACHE_LOCAL	L1 DC fills from local Memory 同じパッケージ (ノード) 内で接続された DRAM または IO からの DC フィル。
0x44、0x04	L1_DC_REFILLS_EXTERNAL_CACHE_LOCAL	L1 DC fills from local external CCX caches 同じパッケージ (ノード) 内の異なる CCX のキャッシュからの DC フィル。
0x44、0x10	L1_DC_REFILLS_EXTERNAL_CACHE_REMOTE	L1 DC fills from remote external CCX caches 異なるパッケージ (ノード) に含まれる CCX キャッシュからの DC フィル。
0x44、0x40	L1_DC_REFILLS_REMOTE_DRAM	L1 DC fills from remote Memory 異なるパッケージ (ノード) 内で接続された DRAM または IO からの DC フィル。
0x44、0x14	L1_DC_REFILLS_EXTENAL_CACHE	L1 DC fills from local external CCX caches 同じまたは異なるパッケージ (ノード) 内の異なる CCX のキャッシュからの DC フィル。
0x44、0x48	L1_DC_REFILLS_DRAM	L1 DC fills from local Memory 同じまたは異なるパッケージ (ノード) 内で接続された DRAM または IO からの DC フィル。
0x44、0x50	L1_DC_REFILLS_REMOTE_NODE	L1 DC fills from remote node 異なるパッケージ (ノード) 内の CCX キャッシュ、または異なるパッケージ (ノード) 内で接続される DRAM/IO からの DC フィル。
0x44、0x03	L1_DC_REFILLS_LOCAL_CACHE_L2_L3	L1 DC fills from same CCX ローカル L2 キャッシュから、同じ CCX 内のコアあるいは異なる L2 キャッシュ、または同じ CCX に属する L3 キャッシュへの DC フィル。

表 64. 事前定義コア PMC イベント (続き)

イベント ID、 ユニット マスク	イベント 略称	名前と説明
0x44、0xDF	L1_DC_REFILLS_ALL	L1 DC fills from all the data sources すべてのデータ ソースからの DC フィル。
0x60、0xFF	L2_REQUESTS_ALL	All L2 cache requests
0x84、0x00	L1_ITLB_MISSES_L2_HITS	L1 TLB miss L2 TLB hit L1 命令変換ルックアサイド バッファ (ITLB) でミスするが、L2 ITLB でヒットする命令フェッチ。
0x85、0x07	L2_ITLB_MISSES	L1 TLB miss L2 TLB miss ページ テーブル ウォークによる ITLB リロード。 テーブル ウォーク要求の対象は、L1-ITLB ミスと L2-ITLB ミスです。
0x45、0xFF	L1_DTLB_MISSES	L1 DTLB miss ロード/ストアのマイクロオペレーションによりミスする L1 データ変換ルックアサイド バッファ (DTLB)。 このイベントでは、L2-DTLB ヒットと L2-DTLB ミスの両方がカウントされます。
0x45、0xF0	L2_DTLB_MISSES	L1 DTLB miss ロード/ストアのマイクロオペレーションによりミスした L2 データ変換ルックアサイド バッファ (DTLB)。
0x78、0xFF	ALL_TLB_FLUSHES	All TLB flushes
0x47、0x03	MISALIGNED_LOADS	ミスアライン ロードの数。 注記: AMD「Zen 3」コアプロセッサで、このイベントは 64B (キャッシュライン横断) および 4K (ページ横断) のミスアライン ロードをカウントします。
0x52、0x03	INEFFECTIVE_SW_PF	Ineffective Software Prefetches プロセッサ コアの外部でデータをフェッチしなかったソフトウェアプリフェッチの数。このイベントは、割り当て済みの要求バッファ ミスで一致を見つけたソフトウェア PREFETCH 命令をカウントします。また、DC ヒットを見つけたソフトウェア PREFETCH 命令もカウントします。
0x18E、0x1F	IC_TAG_ALL_IC_ACCESS	IC Tag All Instruction Cache Access
0x18E、0x18	IC_TAG_IC_MISS	IC Tag Instruction Cache Miss

表 64. 事前定義コア PMC イベント (続き)

イベント ID、 ユニットマスク	イベント略称	名前と説明
0x28F、0x07	OP_CACHE_ALL_ACCESS	All OP Cache Accesses
0x28F、0x04	OP_CACHE_MISS	Op Cache Miss

次の表に、CPU パフォーマンス メトリクスを示します。

表 65. コア CPU メトリクス

CPU メトリクス	説明
Core Effective Frequency	サンプリング時間全体にわたる、一時停止サイクルなしのコア有効周波数を、GHz でレポートします。このメトリクスは、APERF および MPERF MSR をベースとします。MPERF は、コアが C0 ステートである間、P0 ステート周波数でコアによってインクリメントされます。コアが C0 ステートのとき、APERF は実際のコア サイクル数に比例してインクリメントします。
IPC	サイクルあたりのリタイア済み命令数 (IPC) は、1 サイクルあたりにリタイアした平均の命令数です。これは、コア PMC イベントである PMCx0C0 [Retired Instructions] と PMCx076 [CPU Clocks not Halted] を使用して測定されます。これらの PMC イベントは、OS とユーザー モードの両方でカウントされます。
CPI	リタイア済み命令あたりのサイクル数 (CPI) は、IPC メトリクスの逆数です。これは、キャッシュミス、分岐予測ミス、メモリレイテンシ、その他のボトルネックにより、アプリケーションの実行にどう影響が及ぶかを示す基本的なパフォーマンスメトリクスの 1 つです。CPI 値が低いほど、パフォーマンスは良くなります。
L1_DC_REFILLS_ALL (PTI)	1000 のリタイア済み命令あたりのデマンド データ キャッシュ (DC) フィル数。これらのデマンド DC フィルは、ローカル L2/L3 キャッシュ、リモート キャッシュ、ローカル メモリ、リモート メモリなど、すべてのデータソースからフィルされたものです。
L1_DC_MISSES (PTI)	1000 のリタイア済み命令あたりの、L1 データ キャッシュ ミスによる L2 キャッシュ アクセス要求数この L2 キャッシュ アクセス要求には、ハードウェアおよびソフトウェアプリフェッチも含まれます。
L1_DC_ACCESS_RATE	DC アクセスレートは、DC アクセス数をリタイア済み命令の合計数で割ったものです。
L1_DC_MISS_RATE	DC ミスレートは、DC ミス数をリタイア済み命令の合計数で割ったものです。
L1_DC_MISS_RATIO	DC ミス率は、DC ミス数を DC アクセスの合計数で割ったものです。
RETIRED_BR_INST_MISP_RATIO	このメトリクスは、予測ミスしたリタイア済み分岐数をリタイア済み分岐命令の合計数で割ったものとして算出されます。
RETIRED_BR_INST_RATE	リタイアした分岐命令のレート。このメトリクスは、リタイア済み分岐数をリタイア済み命令の合計数で割ったものとして算出されます。

表 65. コア CPU メトリクス (続き)

CPU メトリクス	説明
RETIRED_BR_INST_MISP_RATE	このメトリクスは、予測ミスしたリタイア済み分岐数をリタイア済み命令の合計数で割ったものとして算出されます。
RETIRED_TAKEN_BR_INST (PTI)	1000 命令あたりの成立したリタイア済み分岐の数。
RETIRED_TAKEN_BR_INST_RATE	成立したリタイア済み分岐レート。このメトリクスは、成立したリタイア済み分岐数をリタイア済み命令の合計数で割ったものとして算出されます。
RETIRED_TAKEN_BR_INST_MISP (PTI)	1000 命令あたりの、予測ミスし、成立したリタイア済み分岐の数。
RETIRED_INDIRECT_BR_INST_MISP (PTI)	1000 命令あたりの間接的なリタイア済み分岐の数。
RETIRED_NEAR_RETURNS (PTI)	1000 命令あたりのリタイア済み near 分岐の数。
RETIRED_NEAR_RETURNS_MISP (PTI)	1000 命令あたりの、予測ミスしたリタイア済み near 分岐の数。
RETIRED_NEAR_RETURNS_MISP_RATE	このメトリクスは、予測ミスしたリタイア済み near リターン数をリタイア済み命令の合計数で割ったものとして算出されます。
RETIRED_NEAR_RETURNS_MISP_RATIO	このメトリクスは、予測ミスしたリタイア済み near リターン数をリタイア済みリターン命令の合計数で割ったものとして算出されます。
L1_DTLB_MISS_RATE	DTLB L1 ミス レートは、DTLB L1 ミス数をリタイア済み命令の合計数で割ったものです。
L2_DTLB_MISS_RATE	L2 DTLB ミス レートは、L2 DTLB ミス数をリタイア済み命令の合計数で割ったものです。
L1_ITLB_MISS_RATE	ITLB L1 ミス レートは、ITLB L1_Miss_L2_Hits と L1_Miss_L2_Miss を足した数を、リタイア済み命令の合計数で割ったものです。
L2_ITLB_MISS_RATE	ITLB L2 ミス レートは、ITLB L2 ミス数をリタイア済み命令の合計数で割ったものです。
MISALIGNED_LOADS_RATIO	ミスアライン率は、ミスアライン ロード数を DC アクセスの合計数で割ったものです。
MISALIGNED_LOADS_RATE	ミスアライン レートは、ミスアライン ロード数をリタイア済み命令の合計数で割ったものです。
STLI_OTHER	ストア対ロードの競合: 古いストアとの転送不能な競合により、ロードが完了できなかった状態です。最もよくあるのは、ロードのアドレス範囲が、完全にではなく部分的に未完了の古いストアと重複しているケースです。データにアクセスする際、サイズおよびアライメントが一致するロードとストアを使用することで、ソフトウェアによりこの問題を回避できます。 Vector/SIMD コードは特にこの問題の影響を受けやすいため、サイズの小さい要素単位のストアを使用するのではなく、メモリに保存する前に、ソフトウェアで shuffle/blend/swap 命令を使用してレジスタ内のベクター要素を操作することで、サイズの大きいベクター ストアを構築する必要があります。

表 65. コア CPU メトリクス (続き)

CPU メトリクス	説明
L2_CACHE_ACCESSES_FROM_IC_MISSES	1000 のリタイア済み命令あたりの、L1 命令キャッシュ ミスによる L2 キャッシュ アクセス要求数。この L2 キャッシュ アクセス要求には、プリフェッチも含まれます。
L2_CACHE_MISSES_FROM_IC_MISSES	1000 のリタイア済み命令あたりの、L1 命令キャッシュ ミスによる L2 キャッシュ ミス数。

13.2.5 IBS の派生イベント

AMD uProf は、ハードウェアによって生成された IBS 情報を、EBP サンプル カウントに似た派生イベント サンプル カウントに変換します。すべての IBS 派生イベントの名前と略称には IBS が含まれます。IBS の派生イベントおよびサンプル カウントは、EBP イベントおよびサンプル カウントと似ているように見えますが、IBS イベント情報のソースおよびサンプリング基準は異なります。

IBS 派生イベント サンプル数と EBP イベント サンプル数の間での算術計算は、決して実行しないでください。同じハードウェア条件を表すイベントに対して取得されたサンプル数を直接比較しても意味はありません。たとえば、IBS DC ミス サンプル数が少ない場合も、より多い EBP DC ミス サンプル数と比べて必ずしも良いとは限りません。

次の表に、IBS フェッチ イベントを示します。

表 66. IBS フェッチ イベント

IBS フェッチ イベント	説明
AMD “Zen1”、AMD “Zen2”、AMD “Zen3” クライアント プラットフォーム	
IBS_FETCH	すべての IBS フェッチ サンプルの数。この派生イベントは、収集されたすべての IBS フェッチ サンプル数をカウントし、これには IBS によって中止されたフェッチ サンプルが含まれます。
IBS_FETCH_KILLED	IBS によってサンプリングされたフェッチのうち、中止されたフェッチの数。フェッチが ITLB または IC アクセスに達しなかった場合、フェッチ オペレーションは中止されます。一般に、中止されたフェッチ サンプル数は解析には有用でないため、その他の派生 IBS フェッチ イベントでは除外されます (Event Select 0xF000 を除く。これは、IBS によって中止されたフェッチ サンプルを含むすべての IBS フェッチ サンプルをカウントする)。
IBS_FETCH_ATTEMPT	IBS でサンプリングされたフェッチのうち、フェッチ試行が中止されなかったものの数。この派生イベントは、有用なフェッチ試行の数をカウントし、IBS によって中止されたフェッチ サンプル数は含みません。このイベントは、試行されたフェッチに対する IBS フェッチ IC ミス率など、割合の計算に使用します。試行されたフェッチの数は、完了したフェッチ数とアボートされたフェッチ数の合計に等しくなります。
IBS_FETCH_COMP	IBS でサンプリングされたフェッチで完了したものの数。フェッチ試行によって命令データが命令デコーダーに渡されると、フェッチが完了します。命令データが渡された場合でも、使用されない可能性があります。たとえば、誤って予測された分岐の「正しくないパス」に、命令データが含まれていた可能性があります。

表 66. IBS フェッチ イベント (続き)

IBS フェッチ イベント	説明
IBS_FETCH_ABORT	IBS でサンプリングされたフェッチのうち、アボートされたものの数。フェッチ試行が完了せず、デコーダーに命令データを渡さなかった場合、この試行はアボートされています。フェッチ試行は、命令データのフェッチ プロセス中、任意の箇所アボートできます。アボートの原因として、分岐予測ミスによる分岐先の変更があります。IBS のアボートされたフェッチのサンプル数は、失敗した投機的フェッチ アクティビティ数の下限になります。下限となるのは、完了したフェッチによって渡された命令データは使用されない場合があるためです。
IBS_L1_ITLB_HIT	IBS のフェッチ試行サンプルで、フェッチ オペレーションが最初に L1 ITLB (命令変換 ルックアサイド バッファ) でヒットしたものの数。
IBS_ITLB_L1M_L2H	IBS のフェッチ試行サンプルで、フェッチ オペレーションが最初に L1 ITLB でミスし、L2 ITLB でヒットしたものの数。
IBS_ITLB_L1M_L2M	IBS のフェッチ試行サンプルで、フェッチ オペレーションが最初に L1 ITLB と L2 ITLB の両方でミスしたものの数。
IBS_IC_MISS	IBS のフェッチ試行サンプルで、フェッチ オペレーションが最初に IC (命令キャッシュ) でミスしたものの数。
IBS_IC_HIT	IBS のフェッチ試行サンプルで、フェッチ オペレーションが最初に IC でヒットしたものの数。
IBS_4K_PAGE	IBS フェッチ試行サンプルで、フェッチ オペレーションにより、有効な物理アドレス (アドレス変換が正しく完了したもの) が生成され、L1 ITLB で 4 KB ページ エントリが使用された数。
IBS_2M_PAGE	IBS フェッチ試行サンプルで、フェッチ オペレーションにより、有効な物理アドレス (アドレス変換が正しく完了したもの) が生成され、L1 ITLB で 2 MB ページ エントリが使用された数。
IBS_FETCH_LAT	すべての IBS フェッチ試行サンプルの合計レイテンシ。IBS フェッチ レイテンシの合計を、IBS フェッチ試行サンプル数で割ることで、サンプリングされたフェッチ試行の平均レイテンシが得られます。
IBS_FETCH_L2C_MISS	L2 キャッシュでミスした命令フェッチ。
IBS_ITLB_REFILL_LAT	サンプリングされたフェッチの ITLB リロードのためにフェッチ エンジンがストールしていた間のサイクル数。リロードが実行されない場合のレイテンシは 0 になります。
AMD “Zen3” および AMD “Zen4” サーバー プラットフォーム	
IBS_FETCH	すべての IBS フェッチ サンプルの数。この派生イベントは、収集されたすべての IBS フェッチ サンプル数をカウントし、これには IBS によって中止されたフェッチ サンプルが含まれます。
IBS_FETCH_ATTEMPTED	IBS でサンプリングされたフェッチのうち、フェッチ試行が中止されなかったものの数。この派生イベントは、有用なフェッチ試行の数をカウントし、IBS によって中止されたフェッチ サンプル数は含みません。このイベントは、試行されたフェッチに対する IBS フェッチ IC ミス率など、割合の計算に使用します。試行されたフェッチの数は、完了したフェッチ数とアボートされたフェッチ数の合計に等しくなります。

表 66. IBS フェッチ イベント (続き)

IBS フェッチ イベント	説明
IBS_FETCH_COMPLETED	IBS でサンプリングされたフェッチのうち、完了したものの数。フェッチ試行によって命令データが命令デコーダーに渡されると、フェッチが完了します。命令データが渡されたが、使用されていない可能性があります (たとえば、誤って予測された分岐の正しくないパスに、命令データが含まれていた可能性があります)。
IBS_FETCH_ABORTED	IBS でサンプリングされたフェッチのうち、アボートされたものの数。フェッチ試行が完了せず、デコーダーに命令データを渡さない場合、この試行はアボートされています。フェッチ試行は、命令データのフェッチ プロセス中、任意の箇所でもアボートできます。アボートの原因には、分岐予測ミスによる分岐先の変更があります。IBS のアボートされたフェッチのサンプル数は、失敗した投機的フェッチ アクティビティ数の下限になります。下限となるのは、完了したフェッチによって渡された命令データは使用されない場合があるためです。
IBS_FETCH_L1_ITLB_HIT	IBS のフェッチ試行サンプルで、フェッチ オペレーションが最初に L1 ITLB (命令変換 ルックアサイド バッファ) でヒットしたものの数。
IBS_FETCH_L1_ITLB_MISS_L2_ITLB_HIT	IBS のフェッチ試行サンプルで、フェッチ オペレーションが最初に L1 ITLB でミスし、L2 ITLB でヒットしたものの数。
IBS_FETCH_L1_ITLB_MISS_L2_ITLB_MISS	IBS のフェッチ試行サンプルで、フェッチ オペレーションが最初に L1 ITLB と L2 ITLB の両方でミスしたものの数。
IBS_FETCH_L1_IC_MISS	IBS のフェッチ試行サンプルで、フェッチ オペレーションが最初に IC (命令キャッシュ) でミスしたものの数。
IBS_FETCH_L1_IC_HIT	IBS のフェッチ試行サンプルで、フェッチ オペレーションが最初に IC でヒットしたものの数。
IBS_FETCH_L1_ITLB_4K_PAGE	IBS フェッチ試行サンプルで、フェッチ オペレーションにより、有効な物理アドレス (例: アドレス変換が正しく完了したもの) が生成され、L1 ITLB で 4 KB ページ エントリが使用された数。
IBS_FETCH_L1_ITLB_2M_PAGE	IBS フェッチ試行サンプルで、フェッチ オペレーションにより、有効な物理アドレス (例: アドレス変換が正しく完了したもの) が生成され、L1 ITLB で 2 MB ページ エントリが使用された数。
IBS_FETCH_L1_ITLB_1G_PAGE	IBS フェッチ試行サンプルで、フェッチ オペレーションにより、有効な物理アドレス (例: アドレス変換が正しく完了したもの) が生成され、L1 ITLB で 1 GB ページ エントリが使用された数。
IBS_FETCH_LAT	すべての IBS フェッチ試行サンプルの合計レイテンシ。IBS フェッチ レイテンシの合計を、IBS フェッチ試行サンプル数で割ることで、サンプリングされたフェッチ試行の平均レイテンシが得られます。
IBS_FETCH_L2_MISS	L2 キャッシュでミスした命令フェッチ。
IBS_FETCH_ITLB_REFILL_LAT	サンプリングされたフェッチの ITLB リロードのためにフェッチ エンジンがストールしていた間のサイクル数。リロードが実行されない場合のレイテンシは 0 になります。
IBS_FETCH_OP_CACHE_MISS	IBS フェッチ試行サンプルで、オペレーション キャッシュにより、タグ付けされたフェッチ用のバイトの一部が供給できなかったものの数。

表 66. IBS フェッチ イベント (続き)

IBS フェッチ イベント	説明
IBS_FETCH_L3_MISS	IBS のフェッチ試行サンプルで、命令フェッチが同じ CCX 上の L3 キャッシュでミスしたものの数。

次の表に、IBS フェッチ メトリクスを示します。

表 67. IBS フェッチ メトリクス

IBS フェッチ メトリクス	説明
IBS_FETCH_L1_IC_MISS_RATE_%	IBS フェッチ試行の合計数に対する IBS フェッチ L1 命令キャッシュ ミスの割合 (%)。
IBS_FETCH_LAT_AVE	平均 IBS フェッチ レイテンシ。IBS フェッチ レイテンシを IBS フェッチ試行の合計数で割って算出します。
IBS_FETCH_L1_ITLB_MISS_L2_ITLB_HIT_RATE_%	IBS フェッチ試行の合計数に対する、IBS フェッチ L1 ITLB ミスおよび L2 ITLB ヒットの割合 (%)。
IBS_FETCH_L1_ITLB_MISS_L2_ITLB_MISS_RATE_%	IBS フェッチ試行の合計数に対する、IBS フェッチ L1 および L2 ITLB ミスの割合 (%)。

次の表に、IBS オペレーション イベントの一覧を示します。

表 68. IBS オペレーション イベント

IBS オペレーション イベント	説明
AMD “Zen1”、“Zen2”、“Zen3” クライアント プラットフォーム	
IBS_ALL_OPS	収集されたすべての IBS オペレーション サンプルの数。これらのオペレーション サンプルには、分岐オペレーション、再同期オペレーション、ロード/ストアオペレーション、または区別されないオペレーション (例: 算術演算、論理演算などを実行するオペレーション) があります。IBS により、リタイアしたオペレーションのデータが収集されます。パイプラインフラッシュなどの理由により、アボートされたオペレーションのデータは収集されません。このため、サンプリングされるすべてのオペレーションは、アーキテクチャ上重要であり、正しいプログラム実行に貢献します。
IBS_TAG_TO_RET	すべての IBS オペレーション サンプルでの、タグ付けからリタイアまでのサイクルの合計数。オペレーションのタグ付けからリタイアまでの時間は、オペレーションがタグ付け (サンプリング対象として選択) されてから、そのオペレーションがリタイアするまでのサイクル数で表されます。
IBS_COMP_TO_RET	すべての IBS オペレーション サンプルでの、完了からリタイアまでのサイクルの合計数。オペレーションの完了からリタイアまでの時間は、オペレーションが完了してから、そのオペレーションがリタイアするまでのサイクル数で表されます。
IBS_BR	リタイアした分岐オペレーションの IBS サンプル数。分岐オペレーションは、プログラム制御フローにおける変更であり、無条件分岐と条件付き分岐、サブルーチンの呼び出し、サブルーチンからのリターンを含みます。分岐オペレーションは、AMD64 分岐セマンティクスを実装するために使用されます。

表 68. IBS オペレーション イベント (続き)

IBS オペレーション イベント	説明
IBS_MISP_BR	予測ミスしたリタイア済み分岐オペレーションの IBS サンプル数。このイベントは、すべての分岐オペレーションに対する予測ミスした分岐オペレーションの割合を算出するために使用します。
IBS_TAKEN_BR	分岐成立したリタイア済み分岐オペレーションの IBS サンプル数。
IBS_MISP_TAKEN_BR	予測ミスし、分岐成立したリタイア済み分岐オペレーションの IBS サンプル数。
IBS_RET	オペレーションがサブルーチンのリターンであった、リタイア済み分岐オペレーションの IBS サンプル数。これらのサンプルは、すべてのリタイア済み分岐オペレーションの IBS サンプルに含まれるサブセットです。
IBS_MISP_RET	オペレーションが予測ミスしたサブルーチンのリターンであった、リタイア済み分岐オペレーションの IBS サンプル数。このイベントは、すべてのサブルーチンリターンに対する予測ミスしたリターンの割合を算出するために使用します。
IBS_RESYNC	再同期オペレーションの IBS サンプル数。再同期オペレーションは特定のマイクロコード AMD64 命令のみで見られ、完全なパイプラインフラッシュを引き起こします。
IBS_LOAD_STORE	ロードおよび/またはストア オペレーションを実行するオペレーションの IBS オペレーション サンプル数。各オペレーションが、ロード オペレーションまたはストア オペレーションのいずれか、あるいはその両方を実行します (それぞれが同じアドレスに対して)。
IBS_LOAD	ロード オペレーションを実行するオペレーションの IBS オペレーション サンプル数。
IBS_STORE	ストア オペレーションを実行するオペレーションの IBS オペレーション サンプル数。
IBS_L1_DTLB_HIT	ロードまたはストア オペレーションが最初に L1 DTLB (データ変換ルックアサイド バッファ) でヒットした IBS オペレーションのサンプル数。
IBS_DTLB_L1M_L2H	ロードまたはストア オペレーションが最初に L1 DTLB でミスし、L2 DTLB でヒットした IBS オペレーションのサンプル数。
IBS_DTLB_L1M_L2M	ロードまたはストア オペレーションが最初に L1 DTLB と L2 DTLB の両方でミスした IBS オペレーションのサンプル数。
IBS_DC_MISS	ロードまたはストア オペレーションが最初に L1 DC でミスした IBS オペレーションのサンプル数。
IBS_DC_HIT	ロードまたはストア オペレーションが最初に L1 DC でヒットした IBS オペレーションのサンプル数。
IBS_MISALIGN_ACC	ロードまたはストア オペレーションがミスアラインアクセスを引き起こした IBS オペレーションのサンプル数 (例: ロードまたはストア オペレーションが 128 ビット境界を越えた場合)。
IBS_BANK_CONF_LOAD	ロードまたはストア オペレーションがロード オペレーションとのバンク競合を引き起こした IBS オペレーションのサンプル数。

表 68. IBS オペレーション イベント (続き)

IBS オペレーション イベント	説明
IBS_BANK_CONF_STORE	ロードまたはストア オペレーションがストア オペレーションとのバンク競合を引き起こした IBS オペレーションのサンプル数。
IBS_FORWARDED	ロード オペレーションのデータがストア オペレーションから転送された IBS オペレーションのサンプル数。
IBS_STLF_CANCELLED	ストアからロード オペレーションへのデータ転送が取り消された IBS オペレーションのサンプル数。
IBS_UC_MEM_ACC	ロードまたはストア オペレーションがキャッシュ不可能 (UC) メモリにアクセスした IBS オペレーションのサンプル数。
IBS_WC_MEM_ACC	ロードまたはストア オペレーションが Write Combining (WC) メモリにアクセスした IBS オペレーションのサンプル数。
IBS_LOCKED_OP	ロードまたはストア オペレーションがロックされたオペレーションであった IBS オペレーションのサンプル数。
IBS_MAB_HIT	ロードまたはストア オペレーションが、Miss Address Buffer (MAB) 内の割り当て済みエントリをヒットした IBS オペレーションのサンプル数。
IBS_L1_DTLB_4K	ロードまたはストア オペレーションが有効なリニア (仮想) アドレスを生成し、L1 DTLB 内の 4 KB ページ エントリがアドレス変換に使用された IBS オペレーションのサンプル数。
IBS_L1_DTLB_2M	ロードまたはストア オペレーションが有効なリニア (仮想) アドレスを生成し、L1 DTLB 内の 2 M ページ エントリがアドレス変換に使用された IBS オペレーションのサンプル数。
IBS_L1_DTLB_1G	ロードまたはストア オペレーションが有効なリニア (仮想) アドレスを生成し、L1 DTLB 内の 1 GB ページ エントリがアドレス変換に使用された IBS オペレーションのサンプル数。
IBS_L2_DTLB_4K	ロードまたはストア オペレーションが有効なリニア (仮想) アドレスを生成し、L2 DTLB でヒットし、4 KB ページ エントリがアドレス変換に使用された IBS オペレーションのサンプル数。
IBS_L2_DTLB_2M	ロードまたはストア オペレーションが有効なリニア (仮想) アドレスを生成し、L2 DTLB でヒットし、2 MB ページ エントリがアドレス変換に使用された IBS オペレーションのサンプル数。
IBS_L2_DTLB_1G	ロードまたはストア オペレーションが有効なリニア (仮想) アドレスを生成し、L2 DTLB でヒットし、1 GB ページ エントリがアドレス変換に使用された IBS オペレーションのサンプル数。
IBS_LOAD_DC_MISS_LAT	ロード オペレーションを実行し、データ キャッシュ内でミスしたすべての IBS オペレーションのサンプルにおける、L1 DC ミス ロードの合計レイテンシ (プロセッサ サイクル数)。ミス レイテンシは、L1 データ キャッシュ ミスが検出されてからデータがコアに渡されるまでのクロック サイクル数で表されます。
IBS_LOAD_RESYNC	ロードの再同期。

表 68. IBS オペレーション イベント (続き)

IBS オペレーション イベント	説明
IBS_NB_LOCAL	ロード オペレーションがローカル プロセッサからサービスされた IBS オペレーションのサンプル数。ノースブリッジ IBS データは、L1 データ キャッシュと L2 データ キャッシュの両方でミスするロード オペレーションのみで有効です。ロード オペレーションがキャッシュ ライン境界を横切る場合、IBS データには下位のキャッシュ ラインへのアクセスが反映されます。
IBS_NB_REMOTE	ロード オペレーションがリモート プロセッサからサービスされた IBS オペレーションのサンプル数。
IBS_NB_LOCAL_L3	ロード オペレーションがローカル L3 キャッシュによってサービスされた IBS オペレーションのサンプル数。
IBS_NB_LOCAL_CACHE	メモリ要求を出しているコアのシブリングであるローカルコアに属するキャッシュ (L1 または L2 データ キャッシュ) によって、ロード オペレーションがサービスされた IBS オペレーションのサンプル数。
IBS_NB_REMOTE_CACHE	1 つ以上のコヒーレントな HyperTransport リンクの横断後に、リモート L1 データ キャッシュ、L2 キャッシュ、または L3 キャッシュによってロード オペレーションがサービスされた IBS オペレーションのサンプル数。
IBS_NB_LOCAL_DRAM	ロード オペレーションがローカル システム メモリ (メモリ コントローラー経由のローカル DRAM) によってサービスされた IBS オペレーションのサンプル数。
IBS_NB_REMOTE_DRAM	ロード オペレーションが (1 つ以上のコヒーレントな HyperTransport リンクの横断後にリモート メモリ コントローラー経由で) リモート システム メモリによってサービスされた、IBS オペレーションのサンプル数。
IBS_NB_LOCAL_OTHER	ロード オペレーションがローカル MMIO、設定/PCI スペース、またはローカル APIC からサービスされた IBS オペレーションのサンプル数。
IBS_NB_REMOTE_OTHER	ロード オペレーションがリモート MMIO、設定、または PCI スペースからサービスされた IBS オペレーションのサンプル数。
IBS_NB_CACHE_MODIFIED	ロード オペレーションがローカルまたはリモート キャッシュからサービスされ、キャッシュ ヒット ステートが Modified (M) であった IBS オペレーションのサンプル数。
IBS_NB_CACHE_OWNED	ロード オペレーションがローカルまたはリモート キャッシュからサービスされ、キャッシュ ヒット ステートが Owned (O) であった IBS オペレーションのサンプル数。
IBS_NB_LOCAL_LAT	ローカル プロセッサによってサービスされたロードオペレーションでのデータ キャッシュ ミスの合計レイテンシ (プロセッサ サイクル数)。
IBS_NB_REMOTE_LAT	リモート プロセッサによってサービスされたロードオペレーションでのデータ キャッシュ ミスの合計レイテンシ (プロセッサ サイクル数)。

表 68. IBS オペレーション イベント (続き)

IBS オペレーション イベント	説明
AMD “Zen4” および AMD “Zen3” サーバー プラットフォーム	
IBS_ALL_OPS	収集されたすべての IBS オペレーション サンプルの数。これらのサンプルには、分岐オペレーション、再同期オペレーション、ロード/ストア オペレーションを実行するオペレーション、または区別されないオペレーションがあります。例としては、算術演算、論理演算などを実行するオペレーションがあります。IBS により、リタイアしたオペレーションのデータが収集されます。パイプラインのフラッシュなどの理由により、アボートされたオペレーションのデータは収集されません。このため、サンプリングされるすべてのオペレーションは、アーキテクチャ上重要であり、正しいプログラム実行に貢献します。
IBS_TAG_TO_RET	すべての IBS オペレーション サンプルでの、タグ付けからリタイアまでのサイクルの合計数。オペレーションのタグ付けからリタイアまでの時間は、オペレーションがタグ付け (サンプリング対象として選択) されてから、そのオペレーションがリタイアするまでのサイクル数で表されます。
IBS_COMP_TO_RET	すべての IBS オペレーション サンプルでの、完了からリタイアまでのサイクルの合計数。オペレーションの完了からリタイアまでの時間は、オペレーションが完了してから、そのオペレーションがリタイアするまでのサイクル数で表されます。
IBS_BR	リタイアした分岐オペレーションの IBS サンプル数。分岐オペレーションは、プログラム制御フローにおける変更であり、無条件分岐と条件付き分岐、サブルーチンとサブルーチンからのリターンを含みます。分岐オペレーションは、AMD64 分岐セマンティクスを実装するために使用されます。
IBS_MISP_BR	予測ミスしたリタイア済み分岐オペレーションの IBS サンプル数。このイベントは、すべての分岐オペレーションに対する予測ミスした分岐オペレーションの割合を算出するために使用します。
IBS_TAKEN_BR	分岐成立したリタイア済み分岐オペレーションの IBS サンプル数。
IBS_MISP_TAKEN_BR	予測ミスし、分岐成立したリタイア済み分岐オペレーションの IBS サンプル数。
IBS_RET	オペレーションがサブルーチンのリターンであった、リタイア済み分岐オペレーションの IBS サンプル数。これらのサンプルは、すべてのリタイア済み分岐オペレーションの IBS サンプルに含まれるサブセットです。
IBS_MISP_RET	オペレーションが予測ミスしたサブルーチンのリターンであった、リタイア済み分岐オペレーションの IBS サンプル数。このイベントは、すべてのサブルーチンリターンに対する予測ミスしたリターンの割合を算出するために使用します。
IBS_RESYNC	再同期オペレーションの IBS サンプル数。再同期オペレーションは特定のマイクロコード AMD64 命令のみで見られ、完全なパイプライン フラッシュを引き起こします。
IBS_LOAD_STORE	ロードおよび/またはストア オペレーションを実行するオペレーションの IBS オペレーション サンプル数。各オペレーションが、ロードまたはストア オペレーションのいずれか、あるいはその両方を実行します (それぞれが同じアドレスに対して)。

表 68. IBS オペレーション イベント (続き)

IBS オペレーション イベント	説明
IBS_LOAD	ロード オペレーションを実行するオペレーションの IBS オペレーション サンプル数。
IBS_STORE	ストア オペレーションを実行するオペレーションの IBS オペレーション サンプル数。
IBS_L1_DTLB_HIT	ロードまたはストア オペレーションが最初に L1 DTLB (データ変換ルックアサイド バッファ) でヒットした IBS オペレーションのサンプル数。
IBS_DTLB_L1M_L2H	ロードまたはストア オペレーションが最初に L1 DTLB でミスし、L2 DTLB でヒットした IBS オペレーションのサンプル数。
IBS_DTLB_L1M_L2M	ロードまたはストア オペレーションが最初に L1 DTLB と L2 DTLB の両方でミスした IBS オペレーションのサンプル数。
IBS_DC_MISS	ロードまたはストア オペレーションが最初に L1 データ キャッシュ (DC) でミスした IBS オペレーションのサンプル数。
IBS_DC_HIT	ロードまたはストア オペレーションが最初に L1 データ キャッシュ (DC) でヒットした IBS オペレーションのサンプル数。
IBS_MISALIGN_ACC	ロードまたはストア オペレーションがミスアライン アクセスを引き起こした IBS オペレーションのサンプル数 (ロードまたはストア オペレーションが 128 ビット境界を越えた場合)。
IBS_BANK_CONF_LOAD	ロードまたはストア オペレーションがロード オペレーションとのバンク競合を引き起こした IBS オペレーションのサンプル数。
IBS_BANK_CONF_STORE	ロードまたはストア オペレーションがストア オペレーションとのバンク競合を引き起こした IBS オペレーションのサンプル数。
IBS_FORWARDED	ロード オペレーションのデータがストア オペレーションから転送された IBS オペレーションのサンプル数。
IBS_STLF_CANCELLED	ストアからロード オペレーションへのデータ転送が取り消された IBS オペレーションのサンプル数。
IBS_UC_MEM_ACC	ロードまたはストア オペレーションがキャッシュ不可能 (UC) メモリにアクセスした IBS オペレーションのサンプル数。
IBS_WC_MEM_ACC	ロードまたはストア オペレーションが Write Combining (WC) メモリにアクセスした IBS オペレーションのサンプル数。
IBS_LOCKED_OP	ロードまたはストア オペレーションがロックされたオペレーションであった IBS オペレーションのサンプル数。
IBS_MAB_HIT	ロードまたはストア オペレーションが、Miss Address Buffer (MAB) 内の割り当て済みエントリをヒットした IBS オペレーションのサンプル数。
IBS_L1_DTLB_4K	ロードまたはストア オペレーションが有効なリニア (仮想) アドレスを生成し、L1 DTLB 内の 4 KB ページ エントリがアドレス変換に使用された IBS オペレーションのサンプル数。

表 68. IBS オペレーション イベント (続き)

IBS オペレーション イベント	説明
IBS_L1_DTLB_2M	ロードまたはストア オペレーションが有効なリニア (仮想) アドレスを生成し、L1 DTLB 内の 2 MB ページ エントリがアドレス変換に使用された IBS オペレーションのサンプル数。
IBS_L1_DTLB_1G	ロードまたはストア オペレーションが有効なリニア (仮想) アドレスを生成し、L1 DTLB 内の 1 GB ページ エントリがアドレス変換に使用された IBS オペレーションのサンプル数。
IBS_L2_DTLB_4K	ロードまたはストア オペレーションが有効なリニア (仮想) アドレスを生成し、L2 DTLB でヒットし、4 KB ページ エントリがアドレス変換に使用された IBS オペレーションのサンプル数。
IBS_L2_DTLB_2M	ロードまたはストア オペレーションが有効なリニア (仮想) アドレスを生成し、L2 DTLB でヒットし、2 MB ページ エントリがアドレス変換に使用された IBS オペレーションのサンプル数。
IBS_L2_DTLB_1G	ロードまたはストア オペレーションが有効なリニア (仮想) アドレスを生成し、L2 DTLB でヒットし、1 GB ページ エントリがアドレス変換に使用された IBS オペレーションのサンプル数。
IBS_LOAD_DC_MISS_LAT	ロード オペレーションを実行し、データ キャッシュ内でミスしたすべての IBS オペレーションのサンプルにおける、L1 DC ミス ロードの合計レイテンシ (プロセッサ サイクル数)。ミス レイテンシは、L1 データ キャッシュ ミスが検出されてからデータがコアに渡されるまでのクロック サイクル数で表されます。
IBS_LOAD_RESYNC	ロードの再同期。
IBS_NB_LOCAL	ロード オペレーションがローカル プロセッサからサービスされた IBS オペレーションのサンプル数。ノースブリッジ IBS データは、L1 および L2 データ キャッシュの両方でミスするロード オペレーションのみで有効です。ロード オペレーションがキャッシュ ライン境界を横切る場合、IBS データには下位のキャッシュ ラインへのアクセスが反映されます。
IBS_NB_REMOTE	ロード オペレーションがリモート プロセッサからサービスされた IBS オペレーションのサンプル数。
IBS_NB_LOCAL_L3	ロード オペレーションがローカル L3 キャッシュによってサービスされた IBS オペレーションのサンプル数。
IBS_NB_LOCAL_CACHE	メモリ要求を出しているコアのシブリングであるローカル コアに属するキャッシュ (L1 データ キャッシュまたは L2 キャッシュ) によって、ロード オペレーションがサービスされた IBS オペレーションのサンプル数。
IBS_NB_REMOTE_CACHE	1 つ以上のコヒーレントな HyperTransport リンクの横断後に、リモート L1 データ、L2、または L3 キャッシュによってロード オペレーションがサービスされた IBS オペレーションのサンプル数。
IBS_NB_LOCAL_DRAM	ロード オペレーションがローカル システム メモリ (メモリ コントローラー経由のローカル DRAM) によってサービスされた IBS オペレーションのサンプル数。

表 68. IBS オペレーション イベント (続き)

IBS オペレーション イベント	説明
IBS_NB_REMOTE_DRAM	ロード オペレーションが (1 つ以上のコヒーレントな HyperTransport リンクの横断後にリモート メモリ コントローラー経由で) リモート システム メモリによってサービスされた、IBS オペレーションのサンプル数。
IBS_NB_LOCAL_OTHER	ロード オペレーションがローカル MMIO、設定、PCI スペース、またはローカル APIC からサービスされた IBS オペレーションのサンプル数。
IBS_NB_REMOTE_OTHER	ロード オペレーションがリモート MMIO、設定、または PCI スペースからサービスされた IBS オペレーションのサンプル数。
IBS_NB_CACHE_MODIFIED	ロード オペレーションがローカルまたはリモート キャッシュからサービスされ、キャッシュ ヒット ステートが Modified (M) であった IBS オペレーションのサンプル数。
IBS_NB_CACHE_OWNED	ロード オペレーションがローカルまたはリモート キャッシュからサービスされ、キャッシュ ヒット ステートが Owned (O) であった IBS オペレーションのサンプル数。
IBS_NB_LOCAL_LAT	ローカル プロセッサによってサービスされたロードオペレーションでのデータ キャッシュ ミスの合計レイテンシ (プロセッサ サイクル数)。
IBS_NB_REMOTE_LAT	リモート プロセッサによってサービスされたロードオペレーションでのデータ キャッシュ ミスの合計レイテンシ (プロセッサ サイクル数)。

次の表に、AMD “Zen4” および AMD “Zen3” サーバー プラットフォームでの IBS オペレーション メトリクス一覧を示します。

表 69. AMD “Zen4” および AMD “Zen3” サーバー プラットフォームでの IBS オペレーション メトリクス

IBS オペレーション メトリクス	説明
%IBS_BR_TAG_TO_RETIRE_CYCLES	リタイア済みサイクルに対する IBS 分岐オペレーションのタグ付けの割合。
%IBS_BR_MISP_TAG_TO_RETIRE_CYCLES	リタイア済みサイクルに対する IBS 分岐予測ミス オペレーションのタグ付けの割合。
%IBS_TAKEN_BR_TAG_TO_RETIRE_CYCLES	リタイア済みサイクルに対する IBS 成立分岐オペレーションのタグ付けの割合。
%IBS_RET_TAG_TO_RETIRE_CYCLES	リタイア済みサイクルに対する IBS 分岐リターン オペレーションのタグ付けの割合。
%IBS_BR_COMP_TO_RETIRE_CYCLES	リタイア済みサイクルに対する IBS 分岐オペレーションの完了の割合。
%IBS_BR_MISP_COMP_TO_RETIRE_CYCLES	リタイア済みサイクルに対する IBS 分岐予測ミス オペレーションの完了の割合。
%IBS_TAKEN_BR_COMP_TO_RETIRE_CYCLES	リタイア済みサイクルに対する IBS 成立分岐オペレーションの完了の割合。
%IBS_RET_COMP_TO_RETIRE_CYCLES	リタイア済みサイクルに対する IBS 分岐リターン オペレーションの完了の割合。

表 69. AMD “Zen4” および AMD “Zen3” サーバー プラットフォームでの IBS オペレーション メトリクス (続き)

IBS オペレーション メトリクス	説明
IBS_BR_MISP_RATE_%	分岐予測ミス レート (%). 分岐予測ミス数を分岐オペレーションの合計数で割ったものを % で表示。
%IBS_L1_DTLB_REFILL_LAT_CYCLES	L1 DTLB ミスのために浪費されたサイクルの割合。L1 DTLB リフィル レイテンシのサイクル数を、すべてのオペレーションのタグ付けからリタイアまでの合計サイクル数で割ったものを % で表示。
IBS_ST_L1_DC_MISS_RATE_%	ストアの L1 DC ミス レート (%). スタアの L1 DC ミス数をストア オペレーションの合計数で割ったものを % で表示。
IBS_LD_L1_DC_MISS_RATE_%	ロードの L1 DC ミス レート (%). ロードの L1 DC ミス数をロード オペレーションの合計数で割ったものを % で表示。
IBS_LD_L1_DC_HIT_RATE_%	ロードの L1 DC ヒット レート (%). ロードの L1 DC ヒット数をロード オペレーションの合計数で割ったものを % で表示。
IBS_LD_L2_HIT_RATE_%	ロードの L2 ヒット レート (%). ロードの L2 ヒット数をロード オペレーションの合計数で割ったものを % で表示。
IBS_LD_LOCAL_CACHE_HIT_RATE_%	ロード オペレーションが、共有 L3 キャッシュまたは同じ CCX 内のその他の L1/L2 キャッシュによってサービスされたロード サンプルの割合。IBS_LD_LOCAL_CACHE_HIT を IBS_LOAD で割った数値を % で表示。
IBS_LD_PEER_CACHE_HIT_RATE_%	同じ NUMA ノードの異なる CCX に含まれる L2/L3 キャッシュによって、ロード オペレーションがサービスされたロード サンプルの割合。IBS_LD_PEER_CACHE_HIT を IBS_LOAD で割った数値を % で表示。
IBS_LD_RMT_CACHE_HIT_RATE_%	異なる NUMA ノードの異なる CCX に含まれる L2/L3 キャッシュによって、ロード オペレーションがサービスされたロード サンプルの割合。IBS_LD_RMT_CACHE_HIT を IBS_LOAD で割った数値を % で表示。
IBS_LD_LOCAL_DRAM_HIT_RATE_%	同じ NUMA ノードのローカル システム メモリ (メモリ コントローラーを介したローカル DRAM) によって、ロード オペレーションがサービスされたロード サンプルの割合。IBS_LD_LOCAL_DRAM_HIT を IBS_LOAD で割った数値を % で表示。
IBS_LD_RMT_DRAM_HIT_RATE_%	異なる NUMA ノード内の DRAM によって、ロード オペレーションがサービスされたロード サンプルの割合。IBS_LD_RMT_DRAM_HIT を IBS_LOAD で割った数値を % で表示。

表 69. AMD “Zen4” および AMD “Zen3” サーバー プラットフォームでの IBS オペレーション メトリクス (続き)

IBS オペレーション メトリクス	説明
IBS_LD_DRAM_HIT_RATE_%	システム内の DRAM によって、ロード オペレーションがサービスされたロード サンプルの割合。 IBS_LD_DRAM_HIT を IBS_LOAD で割った数値を % で表示。
IBS_LD_NVDIMM_HIT_RATE_%	システム内の NVDIMM によって、ロード オペレーションがサービスされたロード サンプルの割合。 IBS_LD_NVDIMM_HIT を IBS_LOAD で割った数値を % で表示。
IBS_LD_EXT_MEM_HIT_RATE_%	システム内の拡張メモリによって、ロード オペレーションがサービスされたロード サンプルの割合。 IBS_LD_EXT_MEM_HIT を IBS_LOAD で割った数値を % で表示。
IBS_LD_PEER_AGENT_MEM_RATE_%	システム内のピア エージェント メモリによって、ロード オペレーションがサービスされたロード サンプルの割合。 IBS_LD_EXT_MEM_HIT を IBS_LOAD で割った数値を % で表示。
IBS_LD_NON_MAIN_MEM_HIT_RATE_%	ロード オペレーションが MMIO、設定/PCI スペース、またはシステム内のローカル APIC からサービスされたロード サンプルの割合。IBS_LD_NON_MAIN_MEM_HIT を IBS_LOAD で割った数値を % で表示。
IBS_LD_L1_DC_MISS_LAT_AVE	平均ロード L1 DC ミス レイテンシ サイクル。ロードの L1 DC ミス レイテンシの数値をロードの L1 DC ミス レイテンシの合計サイクル数で割ったもの。
%IBS_LD_L1_DC_MISS_LAT_CYCLES	データのフェッチに浪費されたサイクルの割合。ロード L1 DC ミス レイテンシのサイクル数を、すべてのオペレーションのタグ付けからリタイアまでの合計サイクル数で割ったものを % で表示。
%IBS_LD_L2_HIT_LAT	ロード L1 DC ミス レイテンシ サイクルに対する、IBS ロード L2 ヒット レイテンシ サイクルの割合。
%IBS_LD_LOCAL_CACHE_HIT_LAT	ロード L1 DC ミス レイテンシ サイクルに対する、IBS ロード ローカル キャッシュ ヒット レイテンシ サイクルの割合。
%IBS_LD_PEER_CACHE_HIT_LAT	ロード L1 DC ミス レイテンシ サイクルに対する、IBS ロード ピア キャッシュ ヒット レイテンシ サイクルの割合。
%IBS_LD_RMT_CACHE_HIT_LAT	ロード L1 DC ミス レイテンシ サイクルに対する、IBS ロード リモート キャッシュ ヒット レイテンシ サイクルの割合。
%IBS_LD_LOCAL_DRAM_HIT_LAT	ロード L1 DC ミス レイテンシ サイクルに対する、IBS ロード ローカル DRAM ヒット レイテンシ サイクルの割合。
%IBS_LD_RMT_DRAM_HIT_LAT	ロード L1 DC ミス レイテンシ サイクルに対する、IBS ロード リモート DRAM ヒット レイテンシ サイクルの割合。

表 69. AMD “Zen4” および AMD “Zen3” サーバー プラットフォームでの IBS オペレーション メトリクス (続き)

IBS オペレーション メトリクス	説明
%IBS_LD_DRAM_HIT_LAT	ロード L1 DC ミス レイテンシ サイクルに対する、IBS ロード DRAM ヒット レイテンシ サイクルの割合。
%IBS_LD_NVDIMM_HIT_LAT	ロード L1 DC ミス レイテンシ サイクルに対する、IBS ロード NVDIMM ヒット レイテンシ サイクルの割合。
%IBS_LD_EXTN_MEM_HIT_LAT	ロード L1 DC ミス レイテンシ サイクルに対する、IBS ロードの拡張メモリ ヒット レイテンシ サイクルの割合。
%IBS_LD_PEER_AGENT_MEM_HIT_LAT	ロード L1 DC ミス レイテンシ サイクルに対する、IBS ロード ピア エージェント メモリ ヒット レイテンシ サイクルの割合。
%IBS_LD_NON_MAIN_MEM_HIT_LAT	ロード L1 DC ミス レイテンシ サイクルに対する、IBS ロードの非メイン メモリ ヒット レイテンシ サイクルの割合。

13.3 参照用 URL

プロセッサ固有の PMC イベントとその説明については、次の AMD 開発者向けドキュメントを参照してください。

- 『AMD プロセッサ向けプロセッサ プログラミング リファレンス (PPR)』(<https://developer.amd.com/resources/developer-guides-manuals/>)
- 『AMD ファミリ 17h プロセッサ向けソフトウェア最適化ガイド』(https://developer.amd.com/wordpress/media/2013/12/55723_3_00.ZIP)
- 『AMD ファミリ 19h プロセッサ向けソフトウェア最適化ガイド』(<https://www.amd.com/system/files/TechDocs/56665.zip>)