



Help > Introduction

Introduction

Welcome to CodeXL!

CodeXL is a comprehensive tool suite that allows developers to harness the benefits of AMD CPUs, GPUs, and APUs. Its capabilities include:

- Powerful Host and GPU debugging
- · Comprehensive GPU, CPU and Power profiling
- DirectX[®]12 Graphics Frame Analysis
- · Static OpenCL[™] kernel analysis capabilities
- Static OpenGL[®], Vulkan[®] and DirectX[®] shader analysis capabilities

This tool suite enhances accessibility and lets software developers take maximum advantage of heterogeneous computing. **CodeXL** is available as a Visual Studio® extension, as well as a stand-alone user interface application for Windows® operating system and Linux® operating system.

This document is intended for software developers. The chapters on GPU Debugging and Profiling assume a working knowledge of OpenCL and/or OpenGL. The chapter on CPU Profiling assumes an understanding of the concepts of threads and processes, as well as familiarity with CPU architecture. The chapter on Kernel Analysis assumes knowledge of OpenCL and GPU architecture.

Latest Version of This Document

For the latest and greatest version of the documentation, go to the <u>GPUOpen Website</u>.

Contact Information

<u>GPUOpen Website</u> <u>CodeXL GitHub Issues</u> Page

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Getting Started

Getting Started

This section contains information about system requirements, installation, known issues, and support.

- System Requirements
- · Installation Instructions
- Using CodeXL From a Remote Station
- · <u>Known Issues</u>
- · <u>Support</u>

 $[\]ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Getting Started > System Requirements

System Requirements

Operating Systems

- Microsoft Windows 7-64 bit
- Microsoft Windows 8.1-64 bit
- Microsoft Windows 10 64-bit
- Linux 64-bit (Red Hat, Ubuntu, SUSE)

For detailed system requirements see the CodeXL Release Notes in the CodeXL installation folder or on the Documentation section of the <u>CodeXL web page</u>.

CodeXL Visual Studio Extension

- [Optional] Microsoft Visual Studio 2010 (Standard/Professional/Team System Edition)
- [Optional] Microsoft Visual Študio 2012 (Professional/Premium/Ultimate Edition)
- [Optional] Microsoft Visual Studio 2013 (Professional/Premium/Ultimate Edition)
- [Optional] Microsoft Visual Studio 2015 (Professional/Premium/Ultimate Edition)

Profiling OpenCL[™] Applications

- [GPU device] AMD Catalyst driver with OpenCL[™] GPU support
- [GPU device] AMD Radeon[™] HD 5000 series or newer
- AMD APP SDK (<u>requirements</u>)

Power Profiling

• Kaveri, Mullins, Temash or Carrizo APU

For detailed system requirements see the CodeXL Release Notes in the CodeXL installation folder or on the Documentation section of the <u>CodeXL web page</u>.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Getting Started > Installation Instructions

Installation Instructions

- 1. Ensure you have the required components specified in the **System Requirements**.
- 2. Install CodeXL using one of the following methods:

<u>Windows</u>

Run CodeXL_Win_x.x.x.exe

<u>Linux - plain tar</u>

Extract CodeXL tarball using: tar -xvzf CodeXL_Linux_x86_64_x.x.x.tar.gz

On Linux systems, the CodeXL Debian and RPM packages perform the driver installation automatically.

However, if you've downloaded the CodeXL tar archive, you have to install the Power Profiler's Linux driver manually. This includes a simple step of running **<codexl-install-**

dir>/AMDTPwrProfDriverInstall.run with su credentials. Example:

sudo ./AMDTPwrProfDriverInstall.run

<u>Red Hat Linux (including CentOS, Fedora)</u>

Install the CodeXL RPM package using: rpm -Uvh CodeXL_Linux.x.x-0.x86_64.rpm

Ubuntu and other Debian based Linux distributions

Install the CodeXL Debian package using: sudo dpkg -i codexl_x.x.x_64.deb followed by sudo apt-get -f install

Confirm Installation

Assuming you used the default install location, confirm the CodeXL binaries installation with the following steps.

Windows

- The C:\Program Files (C:\Program Files (x86) on 64-bit machines) folder should have a new sub-folder named "CodeXL" (the full path of the CodeXL folder should be: C:\Program Files\CodeXL, or C:\Program Files (x86)\CodeXL on 64-bit machines)
- A CodeXL shortcut has been created (unless you chose not to have shortcut).
- The Control Panel shows CodeXL in the list of installed Programs.

Linux using tarball

- 1. Navigate to <*CodeXL Directory*>
- 2. Launch CodeXL using ./CodeXL. A GUI appears.

Linux RPM / Debian

- 1. Navigate to /opt/AMD/CodeXL_X.Y-ZZZZ/
- 2. Launch CodeXL using ./CodeXL. A GUI appears.

Confirm Visual Studio Extension

For Windows, to confirm that the Visual Studio Extension has been installed successfully:

- 1. Start Microsoft Visual Studio.
- 2. Click on Help > About Microsoft Visual Studio from the Visual Studio main menu bar.
- 3. Under Installed products:, find CodeXL x.x.





Help > Getting Started > Using CodeXL From a Remote Station

Using CodeXL From a Remote Station

CodeXL can be used remotely using the following methods:

- 1. Desktop sharing
- 2. CodeXL Remote Agent

These methods are detailed below.

Desktop Sharing

Run CodeXL graphic client application on the target platform via desktop sharing such as Windows Remote Desktop, VNC, SSH and X forwardng, etc.

Limitations:

- 1. OpenGL applications that are run for debugging/profiling may not recognize the shared desktop as supporting their GL requirements. If this happens run CodeXL locally on the target platform without desktop sharing or use the CodeXL Remote Agent (see below).
- 2. When CodeXL is run remotely using Linux SSH access, CodeXL requires X streaming such as provided by applications like Xming, etc.

CodeXL Remote Agent

Run the CodeXL Remote Agent on the target platform, and run the CodeXL graphic client application on a local station. For more details see **Remote GPU Profiling**, **Power Profiling and GPU Debugging**.

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Getting Started > Known Issues

Known Issues

Check known issues in the CodeXL Release Notes that are found in CodeXL installation directory, and at the <u>CodeXL</u> <u>Forum</u>.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Getting Started > Support

Support

- <u>CodeXL Project in github</u>
- OpenCL Zone
- <u>AMD Accelerated Parallel Processing OpenCL</u> <u>Programming Guide</u>

Report a specific problem or request help for CodeXL on the <u>CodeXL Forum</u>.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL

Using CodeXL

CodeXL contains the following modules:

- General GUI Controls
- Frame Analysis
- GPU Debugger
- CPU Profiler
- GPU Profiler
- Static Analyzer
- Power Profiler
- <u>Remote Frame Analysis, GPU Profiling, Power</u>
 <u>Profiling and GPU Debugging</u>

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > General GUI Controls

General GUI Controls

Several CodeXL views and controls are used by the debugger, the Profiler, and the static analyzer.

- CodeXL Welcome Page
- Getting Started Dialog
- Project Settings
- **Execution Toolbar**
- Properties View
- CodeXL Explorer
- Global Settings dialog
- System Information Dialog
- CodeXL Search Toolbar

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > General GUI Controls > CodeXL Welcome Page

CodeXL Welcome Page

CodeXL Welcome Page lets you access or create projects. It also contains links to common tasks you can be interested in.

The Welcome Page is opened in CodeXL startup. To access the Welcome Page, on the View menu, click Start Page.

The Welcome Page is divided into three main sections:

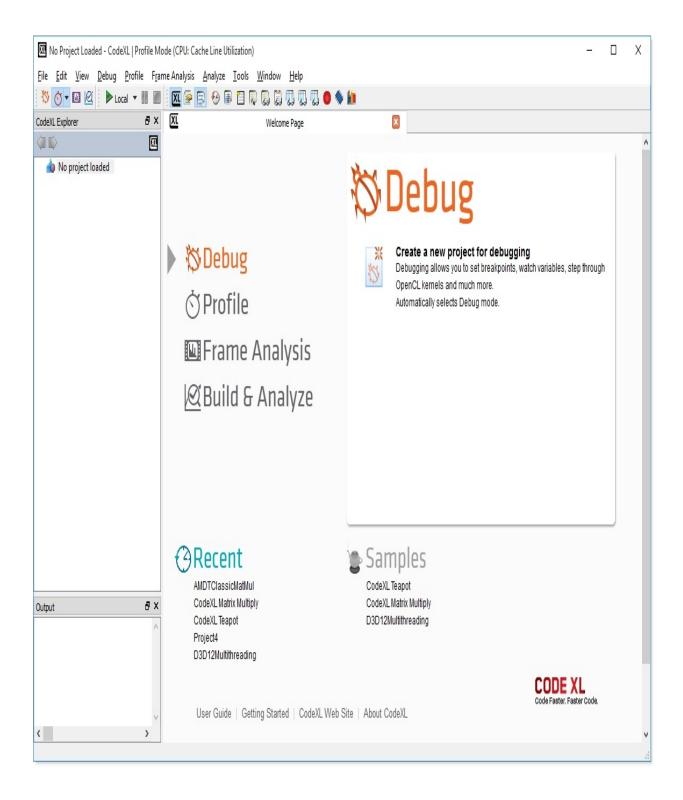
Modes menu: Browse CodeXL modes, and select a task for quick project creation:

	Use this option to greate a new CodeVI
	Use this option to create a new CodeXL
Create New	project. After choosing this option,
Project for	CodeXL will select "Debug Mode" in the
Debugging	execution mode toolbar, and will prepare
	the project for a debugging session.
	Use this option to create a new CodeXL
Create New	project. After choosing this option,
Project for	CodeXL will select "Profile Mode" in the
Profiling	execution mode toolbar, and will prepare
	the project for a profiling session.
	Select this option when you want to
Attach to Process	perform a CPU profiling session on an
	existing process.
	Use this option when you want to
System-wide	perform a CPU or power system-wide
Profiling	profiling session on the currently
	running processes on your machine.
	Selecting this option will create an
	empty new OpenCL kernel file, with a
Create a new	default name. After editing the new
Create a new	

OpenCL file for Analysis	create kernel name, CodeXL will allow you to paste code or edit the kernel code and then build and analyze this code on selected devices.
Add an existing OpenCL file for Analysis	Select this to add an existing OpenCL kernel file and use CodeXL to build and analyze the kernel code on selected devices.

Recent projects: The list displays projects you have worked on recently. Clicking on one of the links will open the project for work.

Samples: Click on CodeXL Teapot sample link, to open the sample project. The sample can be used for getting to know CodeXL capabilities.



 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.



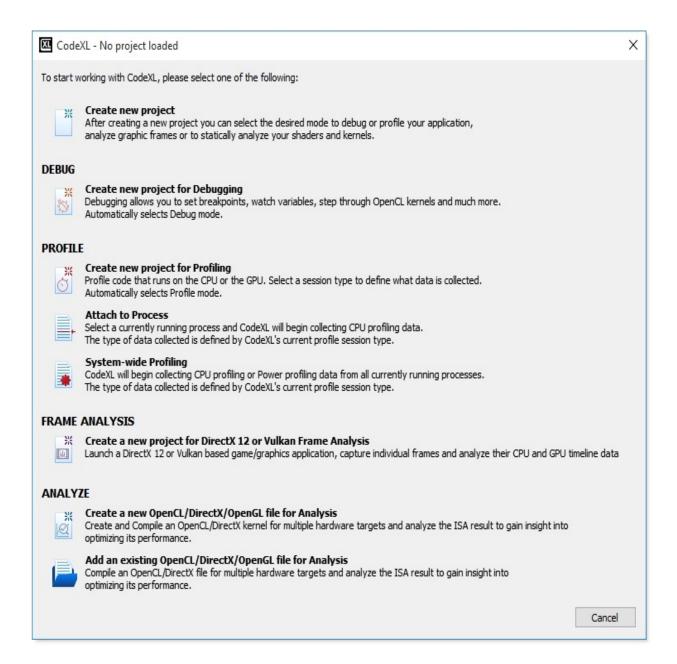


Help > Using CodeXL > General GUI Controls > Getting Started Dialog

Getting Started Dialog

Another way to get start with CodeXL is using the startup dialog. Click the "Start" button (see the following screenshot). A dialog specifying the different options for getting started with CodeXL opens.





Create New Project	Use this option to create a new CodeXL project. The project settings dialog will open and guide you through the steps to create a project with your desired settings.
Create New	Use this option to create a new CodeXL project. After choosing this option,

Project for Debugging	CodeXL will select "Debug Mode" in the execution mode toolbar, and will prepare the project for a debugging session.
Create New Project for Profiling	Use this option to create a new CodeXL project. After choosing this option, CodeXL will select "Profile Mode" in the execution mode toolbar, and will prepare the project for a profiling session.
Attach to Process	Select this option when you want to perform a CPU profiling session on an existing process.
System-wide Profiling	Use this option when you want to perform a CPU or power system-wide profiling session on the currently running processes on your machine.
Create a new OpenCL file for Analysis	Selecting this option will create an empty new OpenCL kernel file, with a default name. After editing the new create kernel name, CodeXL will allow you to paste code or edit the kernel code and then build and analyze this code on selected devices.
Add an existing OpenCL file for Analysis	Select this to add an existing OpenCL kernel file and use CodeXL to build and analyze the kernel code on selected devices.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > General GUI Controls > Project Settings

Project Settings

To start debugging or profiling an application:

1. Create a CodeXL project.

A CodeXL project consists of general information for the debugged / profiled application, such as command-line arguments, environment variables. The project also configures debugging and profiling specific configurations.

 Use the File > New Project menu to open the new project dialog.

The same dialog can be used later to configure the project settings during the debugging / profiling. (Debug > Debug Settings and Profile > Profile Settings menus)

CodeXL Project Settings		Х
General	General	
Debug Y Profile	CodeXL Project Name:	CodeXL Teapot
CPU Profile Application Timeline Trace GPU Profile: Perf. Counters		Local Host Port: 27015 Test Connection
Power Profile	Target Application	
Frame Analysis		Desktop Application Windows Store App
 Analyze Kernel/Shader Build Options 	Executable Path:	
HLSL Build Options		C: \Users \Public \Documents \CodeXL \examples \Teapot \Release \AMDTTeaPot.exe
a de la construction de	Working Directory:	C: \Users\Public\Documents\CodeXL\examples\Teapot\Release
	Command Line Arguments:	
	Environment Variables:	
	Source Files	
	Source Files Directories:	Its\CodeXL\examples\Teapot\AMDTTeapot\src;C:\Users\Public\Documents\CodeXL\examples\Teapot\AMDTTeapotLib;
	Source Files Resolution	
	Root Directory:	
Restore Default Settings		OK Cancel

General Project Settings

CodeXL Project Name	The name of the current project used to identify a project in the Explorer views.		
Target Host	The host on which the executable file will be debugged or profiled (Local / Remote).		
Remote Host Address	The remote host address when the "Remote Host" option is selected.		
Port	The port number on the remote host when the "Remote Host" option is selected.		
Test Connection	Click to test the current connection settings.		
Application Type	The type of application that will be debugged or profiled. The application can be either a desktop application or a Windows Store application.		
Executable Path	The path to the executable / Windows Store application to be debugged or profiled. Use the Browse button for quick selection.		
Working Directory	The directory in which the executable is to be debugged or profiled. Use the Browse button for quick selection.		
Command Line Arguments	Arguments to be passed to the executable.		
Environment Variables	A list of environment variables and their values to be set in the environment for the executable.		
Source Files Directory	List of directories with the source code for the CPU profiled applications or the OpenCL [™] kernels. Use the Browse		

Specific Project Settings

- **<u>GPU Debugging Settings</u>**
- **<u>CPU Profiling Settings</u>**
- **<u>GPU Profiling Settings</u>**

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > General GUI Controls > Execution Toolbar

Execution Toolbar

Once a CodeXL project is started, the initial interface is displayed, as shown in the following screenshot.

🗷 CodeXL Teapot - CodeXL Debug Mode						- 0	Х
<u>File Edit View D</u> ebug <u>P</u> rofile F <u>r</u> ame Anal	ysis <u>A</u> nalyze <u>T</u> ools	<u>W</u> indow <u>H</u> elp					
📉 🔆 🕶 🔟 🖉 🕨 Local 💌 🖩 🞧 🗸	ን ቡ 🕻 🖾 🖗	9 9 8 1 0 0 0		D 🚸 🛍			
CodeXL Explorer 🗗 🛪							
🐞 CodeXL Teapot Debug Mode - Not running							
Properties 🗗 🗙	Function Calls History	₽×	Call Stack	8 :	C Debugged Process Events		đΧ
CodeXL Teapot							
Current Execution Mode: Debug Mode							
To start debugging, select 'Start Debugging' from the Debug menu,							
Debug menu, or dick the 'Start Debugging' toolbar button.							
	Function Calls History	OpenGL State Variables	Call Stack	Breakpoints			
					L		

CodeXL Initial Layout when a project is loaded

The CodeXL toolbar allows quick access to CodeXL views and main controls.

📉 🔆 🕶 🗠 📄 🕨 Local 🔻 🖩 🔳 🥋 ד 🥃 🖙 🦿 🕅 🖾 🛜 🗟 🚱 🗒 🗒 🖏 🥋 🧶 🚺

Debug Mode	3	Switch CodeXL to Debug mode. This is the default mode.
Profile Mode	<u>o</u> -	Switch CodeXL to Profile mode. Use the Profile menu to switch between the various profile modes.
Frame Analysis Mode		Switch CodeXL to Frame Analysis mode.
Analyze Mode	Ø	Switch CodeXL to Analyze mode.
Start Debugging / Profiling	► Local	Start the startup project with CodeXL in the selected mode.
Pause / Stop Debugging / Profiling		Pause / Stop the startup project with CodeXL in the selected mode.
Debug Steps	မ်းနှင့် ရှိ	API step / Draw Step / Frame Step / Step In / Step Over / Step Out the debugged application.
CodeXL Explorer	XL	Show the CodeXL Explorer tree view.
Properties View	@	Show the Properties view.
Output View	B	Show the Output view

Function Calls History View	•	Show the Function Calls History view.
Debugged Process Events View		Show the Debugged Process Events view.
Call Stack View	1	Show the Debugger Call Stack view.
Locals View		Show the Debugger Locals view.
Debugger Watch View	63	Show the Debugger Watch view.
Debugger OpenGL State Variables View		Show the Debugger OpenGL State Variables view.
OpenCL Debugger Multiwatch View 1, 2, 3	IJ	Show the OpenCL Debugger Multiwatch views.
Debugger Breakpoints View	•	Show the Debugger Breakpoints view.
Debugger Memory View	۲	Show the Debugger Memory view.
Debugger Statistics View	in	Show the Debugger Statistics view.

 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > General GUI Controls > Properties View

Properties View

The CodeXL properties view displays detailed information about items appearing in other CodeXL views.

OpenCL Image Properties

- 1. Debug an OpenCL sample.
- 2. Break after clCreateContext.
- 3. Click the OpenCL Context object in the <u>CodeXL</u> <u>Explorer</u> to see the properties view content.

As shown in the image, the properties view specifies the image type, the image handle, dimensions and format.

Properties			x
Texture 1			^
General		V.505 204V	
Туре	2D Texture	uco pur veloperap	
# of Mipmaps	10 Level(s)	000000000	
Dimensions	512x512	initia stato initia stato	
Border Width	0px		
Internal Pixel F	ormat		
Requested	GL_RGBA		
Used	GL_RGBA		~

Clicking on the image thumbnail in the properties view will open an **<u>image view</u>**.

OpenCL Context Properties

- 1. Debug an OpenCL sample with images
- 2. Break the application.
- 3. Click on an OpenCL context in the **<u>CodeXL Explorer</u>** to

see the properties view content.

The properties of an OpenCL context contains the context run time properties.

By clicking the context platform or devices link, the **System Information Dialog** is displayed. This shows platform / devices details.

Properties	
CL Context 1	
General	
Reference Count	35
Marked for Deletion	No
OpenGL Shared Context	<u>GL 1</u>
Context Creation Propert	ties
CL_CONTEXT_PLATFORM	Platform 1
CL_WGL_HDC_KHR	0x59011fa9
CL_GL_CONTEXT_KHR	0x00010000
Context Devices	
Device Id	Device 2
Device Type	CPU
Device Name	AMD FX(tm)-8150 Eight-Core Processor
Device Version	OpenCL 1.2 AMD-APP (1912.5)
Device Vendor	AuthenticAMD
Device Id	Device 1
Device Type	GPU
Device Name	Hawaii
Device Version	OpenCL 1.2 AMD-APP (1912.5)
Device Vendor	Advanced Micro Devices, Inc.

Profile Session

- 1. Set CodeXL execution mode to profile.
- Click on one of the sessions in the <u>CodeXL Explorer</u> to see the properties view content. Session properties view displays the session type, the session file path and working folder, and few of the session properties.

CPU: Profile Session	
Profile Type	Assess Performance
Session Name	Apr-10-2016_11-15-08
Executable Path: C:\Users\Public\	Documents\CodeXL\examples\Teapot\Release\AMDTTeaPot.exe
Working Directory: C:\Users\Public	\Documents\CodeXL\examples\Teapot\Release
Profile Scope: Single Application	
Profile Start Time	Apr-10-2016_11-15-08
Profile End Time	Apr-10-2016_11-15-14
CPU Details: Family 0x15, Model 1	, 8 core(s)
Profile Duration: 5 seconds	
CPU Affinity: 0xff	
Call Stack Information: Scope - Use	er mode, Depth - 32, Frequency - per 1 samples, FPO - Off

© 2015 Advanced Micro Devices. All rights reserved.



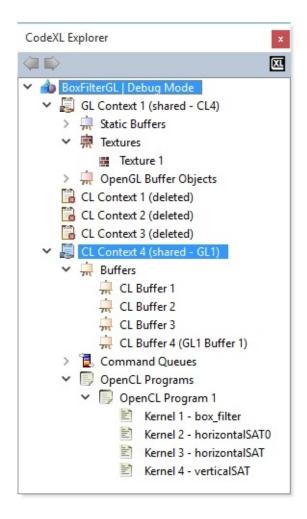


Help > Using CodeXL > General GUI Controls > CodeXL Explorer

CodeXL Explorer

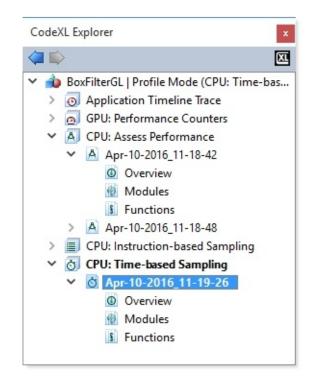
The CodeXL Explorer tree lets you navigate while debugging, profiling, or analyzing.

The tree structure is different for each of the execution modes. In Debug mode, the tree contains the debugged application details **only when the debugged application is paused.** Only one debug tree can be viewed while debugging. In Profile mode, on the other hand, the tree contains all the profiled sessions for the current project. You can view multiple sessions. When switching to Analyze mode, the tree will display all the OpenCL kernel files added to the current project, with build and analyze results for each of them.



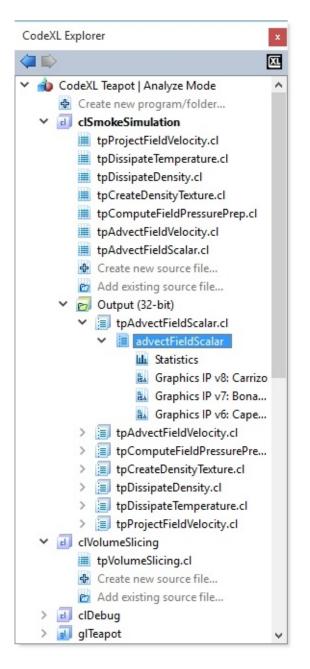
OpenCL / OpenGL Debugged Objects Tree

While debugging, the CodeXL Explorer tree view lets you navigate and access all OpenCL and OpenGL objects during an application run. The Objects tree displays all the debugged application's allocated objects in the hierarchy: Application -> Context -> (Object Type) -> (Specific Object). Contexts marked with the sharing icon have their memory allocated objects shared by at least one other context. See cl_khr_gl_sharing in the *OpenCL Specification*, and wglShareLists in the *OpenGL Specification*, for more details about context resource sharing. Note that sharing contexts can have memory sizes even after being deleted, as long as there is at least one other context sharing their objects.



CodeXL Profile Sessions Tree

In Profile mode, the CodeXL Explorer lists the profiling sessions done for the current project; it indicates the type of profile for each session. When a CodeXL project is opened, the Profiler Session Explorer automatically displays all previous sessions associated with that project. Double-click on a session to open the session data view. Right-click on a session to rename it, delete it or to open the folder that contains the session output file. You can also import a previously-generated Profiler session, or delete all sessions using the context menu in the Profiling Session Explorer. A previously-generated Profiler session can be imported by dragging and dropping the session file from the system file browser to the Session Explorer.



CodeXL Analyze Tree

In Analyze mode, the tree displays 2 sub trees:

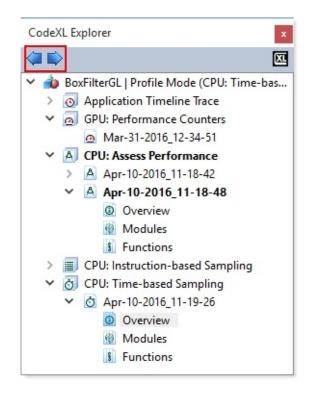
- 1. Tree of programs created by the user, with the output build results.
- Tree of OpenCL / DirectX / OpenGL / Vulkan shader and kernel files added by the user and used while building the programs.

- Double clicking on a source file will open the source file in CodeXL
- Double clicking on an output build results will open open a view that will show the source and IL and/or ISA if applicable for the specific device. The build results are grouped by kernel and then by device families.
- Double-clicking the "Statistics" and "Analysis" nodes will open the appropriate information view.
- You can drag a source file from the tree, in order to place it on a program.

The Tree also allows quick activation of the "Create new source file" and "Add existing source File" commands via doubleclicking the two nodes at the bottom of the tree.

Back / Forward Buttons

The two tree navigation buttons are located in CodeXL Explorer top panel. Use these buttons to navigate to the previously viewed objects in the tree. The navigation history resets when opening another project or a debug session is terminated.

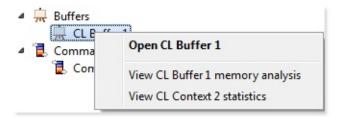


CodeXL Explorer context menu

Use the tree context menu to navigate and manage the current displayed project data.

For example, see the following context menus:

 Context menu on a CL buffer item when the debugged process is paused. The menu will contain the option to display the buffer in an MDI window and to view the buffer memory analysis and statistics.

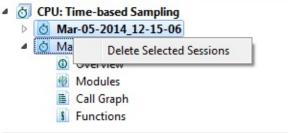


Context menu on a Time-Based CPU profile session. The menu will contain the options to open the session in an

MDI window, rename or delete the session, and open the folder that contains the session.

 CPU: Assess Performance Apr-10-2016_11-18-42 		
Overview	Open Apr-10-2016_11	18-42
Modules	Rename Session	F2
3 Functions	Delete Session	Del
	Open Containing Folder	r
	Import Session	

Context menu on multiple sessions in profile mode. The menu will enable the deletion of multiple sessions.



Context menu on source file. This menu enables one to open the source file, rename it, remove it from the CodeXL project, and open the containing folder.

×

Source Files		
d tpVolumeSlicing.cl	On an AnWalton of Chains	
tpSmokeSimulation	Open tpVolumeSlicing.c	•
🚽 tpProjectFieldVeloci	Rename file	F2
可 tpDissipateTemperat		
tpDissipateDensity.c	Remove from Project	Del
tpDebugVelocityVec	Open Containing Folder	

Context menu on a program. This menu enables one to run the build program command, to rename, remove or change the source of a program.

> clSmokeSimulation > clVolumeSlicing	Build clSmokeSimulation	F7
> 🗐 clDebug	Rename program	F2
> 🗊 glTeapot	Remove clSmokeSimulation from Project	Del
 Source Files tpVolumeSlicing tpSmokeSimulat 	Create a new source file Add existing source file	
tpProjectFieldVel	Export Binaries	





Help > Using CodeXL > General GUI Controls > Global Settings

Global Settings

This dialog box lets you specify global settings for the CodeXL environment.

	GPU Debug	CPU Profile	Analyze			
Logging						
Debug k					INFO) 🔻
_	s and Images Dir					
C:\Use	rs\salgrana\App[Data\Local\Temp	1			
Connect	ion					
I am	using an HTTP p	proxy				
Server:	www.myproxy	.example			Port:	8080 🗘
Demete	Debugging Ports			2023 🗘 2024 🗘	2025 🜲	2026 ≑
Kelliote	Debugging For a			2023 🛡 2024 🛡	2023 🗸	2020 👻
Advance	ed					
Floating	-point precision ((maximum numbe	er of significant d	igits):		8 🜩
Source F	iles					
	t when a source	file is missina				

General Global Settings

Debug Log Level	 The debug log level is an internal mechanism that helps fix CodeXL problems. There are three levels. Error: Log errors that occur while running CodeXL. Info: Log errors and CodeXL internal information seen while running CodeXL (default level). Debug: Log CodeXL debugging information, errors, and other internal information that is generated while running CodeXL. (This can be used by the CodeXL support team to locate a problem inside CodeXL). Extensive: Log extensive CodeXL debugging information, errors, and other internal information that is generated. (This can be used by the CodeXL support team to locate a problem inside CodeXL). Extensive: Log extensive CodeXL debugging information that is generated while running CodeXL. (This can be used by the CodeXL).
Log Files and Images Directory	The directory in which the API call log files and temporary image and source files are saved.
I am using an HTTP proxy server	Check this option if your computer connects to the Internet through an HTTP proxy. If you are experiencing problems with CodeXL's online features (check for updates and send error report), changing the proxy settings might be the solution.
Proxy server / Port number	If you are using a proxy server, please input its information here. The server name could be a DNS address or an IP address. If you are unsure of what your HTTP proxy server or port are, contact your network administrator or copy the settings from your web browser. CodeXL does not currently support the

Remote Debugging Ports	HTTP_PROXY environment variable. Specifies the port numbers that are being used for remote debugging sessions. On the client machine, all four ports should be able to receive incoming connections, and on the remote machine all four ports should not be blocked for outgoing connections.
Floating- point Precision	The display precision for displaying floating- point numbers.
Alert when a source file is missing	Alert the user when a source file is not found during the profile session source code display.
Restore Default Settings	Click this button to restore all settings on all pages to their default values.

Plugin Global Settings

GPU Debugging Global Settings CPU Profiling Global Settings Static Analysis Global Settings





Help > Using CodeXL > General GUI Controls > System Information Dialog

System Information Dialog

The system information dialog displays computer configuration details and OpenCL / OpenGL implementation details. These include operating system, memory, graphic card and driver details; available OpenCL platforms and devices; monitor details, available pixel formats, and available OpenGL extensions.

System	Display	OpenGL Rend	erer	OpenGL Pixel Formats	OpenGL Extensions	OpenCL Platforms	OpenCL Devices	
ltem			Value					^
Computer	Name		DESKT	TOP-04KKOCS				
User			salgra	na				
Operating	System Ve	rsion (name)	Windo	ows 10				
Operating	System Ve	rsion (number)	10.0.1	0240				T
Number o	f CPU Core	25	8					
System Ty	pe		x86					
CPU Famil	У		Family	/ 0x15, Model 0x1, Stepp	ing 0x2			
AMD Rade	on Softwa	re version	15.8					
Total Physi	ical Memo	ry	8137 M	MB				
Available P	hysical Me	emory	4600	ИB				
Total Virtu	al Memory		2047	MB				
Available V	/irtual Men	nory	1405 M	MB				¥
escription	.							
		displays inform	ation ab	out the configuration of y	our system			

System Information Tabs

Collects and displays information about the system configuration.
Collects and displays information about

Display	the display system configuration.
OpenGL Renderer	Collects and displays information about the graphics accelerator configuration.
OpenGL Pixel Formats	Collects and displays information about the system's supported pixel format.
OpenGL Extensions	Collects and displays information about the system's supported OpenGL extensions.
OpenCL Platforms	Collects and displays information about the supported OpenCL platforms.
OpenCL Devices	Collects and displays information about the supported OpenCL devices and their capabilities and limits.
Save Button	Exports the system information data from all views to a .csv file.

OpenCL Devices View

The OpenCL Devices View displays the OpenCL devices available in the 32-bit and 64-bit OpenCL runtimes that are installed on the local station. The information is collected using the modules from the system path and may differ from runtimes installed elsewhere, such as OpenCL modules installed in the AMD APP SDK folder.

CodeXL System Information Display OpenGL Renderer OpenGL Pixel Formats OpenGL Extensions **OpenCL** Platforms OpenCL Devices System Parameter Device 1 (32-bit, GPU) Device 2 (32-bit, CPU) Device 1 (64-bit, GPU) Device 2 (64-bit, (^ Platform ID 1 1 1 1 GPU CPU GPU CPU Device Type Device Name AMD FX(tm)-8150 Ei... Hawaii AMD FX(tm)-8150 Hawaii Vendor Advanced Micro De... AuthenticAMD Advanced Micro De... AuthenticAMD **Command Queue Properties** Queue profiling Queue profiling Queue profiling Queue profiling Max Global Variable Size 0 bytes 1,024 MB 2,592 MB 1,792 MB Global Variable Preferred Total Size 0 bytes 1,024 MB 4,096 MB 1,792 MB Queue On Device Properties N/A N/A Out-of-order executi... N/A Queue On-Device Preferred Size 0 bytes 0 bytes 256 KB 0 bytes **Oueue On-Device Max Size** 0 bvtes 0 bvtes 8 MB 0 bvtes > < Description: OpenCL devices view collects and displays information about the OpenCL devices available on all OpenCL platforms installed on your system. This information is collected using the OpenCL modules in your system path. If the AMD APP SDK is installed it may provide more advanced device capabilities. OK Save

© 2015 Advanced Micro Devices. All rights reserved.

Х





Help > Using CodeXL > General GUI Controls > CodeXL Search Toolbar

CodeXL Search Toolbar

Search toolbar appears when a view with searchable text is in focus and the Ctrl+F shortcut is used or the Find command is clicked in the Edit menu:

🔟 CodeXL Teapot (Kernel Debugging) - CodeXL	Debug N	lode						-		Х
Eile Edit View Debug Profile Frame Ani		nalyze <u>T</u> ools <u>W</u> indow <u>H</u> e) 🗔 🗔 🛛 🚸 🔝	Wavefront 1	▼ Work Item X: 0	v	Y: 0	✓ Z:	0 \
						pos		X v ^	Match <u>C</u> a	ise X
CodeXL Explorer 🖉 🕽	×	amdtteapot-Op	enCLContext1-CLPr	rogram001.d	×					
 CodeXL Teapot Debug Mode GL Context 1 (shared - CL1) GL Context 1 (shared - GL1) GL Context 1 (shared - GL1) Shaders CL Context 1 (shared - GL1) GL Context 1 (shared - GL1) GuencL Program 1 GuencL Program 2 GuencL Program 3 GuencL Program 4 GuencL Program 5 GuencL Program 6 	15 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 30 37 38 30	<pre>// limited // randomix // The velo // to value // a side-u // teapot s // teapot s // teapot s // constant SmokeSimd // constant SmokeSimd</pre>	to be inside ted by the hosport vector i es given by the vise movement spout to the e constants* p) (get_global_id (coord); 2)((float)coor if this grid s - p->spoutCe vtInVExtent.x; vtInVExtent.y; dot(normPos, n	<pre>in the region of sr ne host (generally that points from the exit point.</pre>	<pre>exit point is noke is set acco movement up wit the center of th (scalar) (scalar) ector) d(1), get_global y); ne teaspout, 0.0</pre>	rding h e 				>
Properties		₽ ×	Watch		🗗 🗙 Call Stack		🗗 🗙 Locals			8
Kernel 1 General Kernel Function Name Kernel Handle Reference Count # of Arguments Arguments Information		^	Name Type watch exp	Value (X:0, Y:0, Z:(Ty	rpe <mark>⇒</mark> _Open	CL_applySources_kernel -		Value (X:) el dispa {} 0x000000 0x000000 0x000000 0x000000 0x000000 0x000000 0x000000 viewing c	00 <u>.</u> 00 <u>.</u> 00 .	Type struct _glob _glob _glob _cons
global float*s										
global float* s global float* t		v			Call Stack	Breakpoints	<			

The user can navigate through a view's text content using the Previous and Next buttons. Use the Match Case toggle button to control the case sensitivity of the search. $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Frame Analysis

Frame Analysis

CodeXL Frame Analysis allows the developer to generate an API Trace with CPU side timing data and a GPU Trace with GPU side timing data. Both sets of data are displayed in a unified timeline which quickly allows the user to identify expensive GPU executions and the CPU side API calls which generate them. CodeXL allows you capture one or more frames from your running application and the captures are automatically saved to disk. The captured frames can be inspected offline using the CodeXL client (your application does not need to be running).

There are 4 stages to viewing the API and GPU trace data from your application.

- 1. <u>Creating a New Project</u>
- 2. <u>Starting Your Application</u>
- 3. <u>Capturing Frames</u>
- 4. <u>Viewing Capture Data</u>





Help > Using CodeXL > Frame Analysis > Creating a New Project

Creating a New Project

The CodeXL project allows you to define your application executable, its working directory, and any command line arguments it requires. CodeXL will automatically save your project settings so the next time you want to analyze your application you simply load the project to get started.

- 1. Click "File -> New Project" to open the "Create a new CodeXL Project" dialog.
- 2. Browse to the executable you wish to analyze by clicking on the folder to the right of the "Executable Path:" data entry field.
- 3. The project will automatically be given the same name as the chosen executable. If you wish to use a different project name, enter a new name under the "CodeXL Project Name:" field **after selecting the executable**.
- 4. Enter any command line arguments, with which to run the executable, in the "Command Line Arguments:" data entry field.

Debug Codel ✓ Profile Targe Application Timeline Trace Remo GPU Profile Targe Power Profile Targe Frame Analysis Applic ✓ Analyze Applic > Kernel/Shader Build Options Execution HLSL Build Options Workit	et Host: () ote Host Address: get Application cation Type: () utable Path:	Project1 Local Host Remote Host Port: Port: 27015 Test Desktop Application Windows Store App	Connection
 Profile CPU Profile Application Timeline Trace GPU Profile: Perf. Counters Power Profile Frame Analysis Analyze Kernel/Shader Build Options HLSL Build Options Worki 	et Host: () ote Host Address: get Application cation Type: () utable Path:	Local Host O Remote Host Port: 27015 Test	Connection
Application Timeline Trace GPU Profile: Perf. Counters Power Profile Frame Analysis Analyze Kernel/Shader Build Options HLSL Build Options Worki	ote Host Address: get Application (ation Type: utable Path:	Port: 27015 Test	Connection
Frame Analysis Targ Analyze Applic Kernel/Shader Build Options Execu HLSL Build Options Worki	cation Type:	Desktop Application O Windows Store App	
 ✓ Analyze Applic > Kernel/Shader Build Options Execution HLSL Build Options Workit 	utable Path:	Desktop Application O Windows Store App	
Kernel/Shader Build Options Exect HLSL Build Options Worki			
Work			
Comm	ing Directory:		
	mand Line Arguments:		
Enviro	onment Variables:		
Sour	irce Files		
Source	ce Files Directories:		
	ce Files Resolution Directory:		

Frame Analysis settings

1) In the 'CodeXL Project Settings' dialog or the 'Create a new CodeXL Project' dialog, click on the 'Frame Analysis' tree node to display the 'Frame Analysis Settings'.

- By default the "Automatically connect to first active API" option is turned on. When checked, this setting makes CodeXL automatically connect to the first DX12 application it finds when launching an analysis session. There are a couple circumstances where this behavior is undesired:
 - a. If the executable launches multiple DX12 processes and you want to be able to choose which one to connect to.
 - b. If you have trouble connecting to the DX12 application because CodeXL times out before the program has fully launched.

NOTE: If you have auto-connect disabled, the API selection window will remain open until you select an API context to connect to. Frame capture cannot begin until a connection has been made to a DX12 application.

CodeXL Project Settings		
General	API Selection	
Debug		
✓ Profile	Automatically connect to First active DX12	
> CPU Profile	Server connection port 8080	
Application Timeline Trace		
GPU Profile: Perf. Counters		
Power Profile		
Frame Analysis		
✓ Analyze		
> Kernel/Shader Build Options		
HLSL Build Options		
Cold a Conditional Condition		
Restore Default Settings		OK Cancel



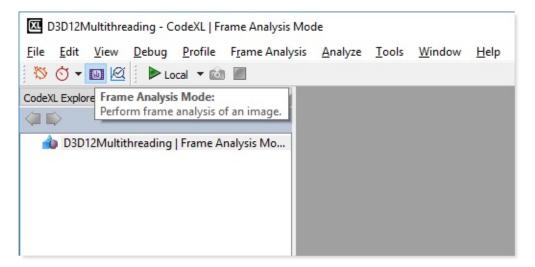


Help > Using CodeXL > Frame Analysis > Starting Your Application

Starting Your Application

1) Switch to Frame Analysis Mode either by clicking the Frame Analysis button

in the toolbar, or by selecting 'Frame Analysis' -> 'Switch to Frame Analysis Mode' from the menu bar.



- 2) Begin a frame capture session either by:
 - a. Clicking the Play button <a>Loca in the toolbar, or
 - b. Selecting 'Frame Analysis' -> 'Start Frame Analysis' from the menu bar, or
 - c. Pressing F5.

🔟 D3D12Multithreading - CodeXL Frame Analysis Mode									
<u>F</u> ile	<u>E</u> dit	<u>V</u> iew	<u>D</u> ebug	<u>P</u> rofile	Frame Analysis	<u>Analyze</u>	<u>T</u> ools	Window	<u>H</u> elp
Ø	Ó •	u Ø	Lo	cal 🔻 🔯					
CodeX	CodeXL Explorer Start Frame Analysis (F5)								
	D3D	12Multit	hreading	Frame A	nalysis Mo				

3) At this point, your application will launch and a selection box will appear with a list of all the API contexts found so far.

E Frame Analysis API Selection	n					?	×
Please select an API to connect to	o:						
# Process 1 D3D12Multithreading.exe	PID 7800	API DX12	Stat Atta	ached	Time lo.	aded	
Monitoring	ession to	First active [API]		•			
				Co	onnect	Cano	el

The above image shows the connection dialog with autoconnect disabled. Auto connect will be enabled by default.

Once a connection is established to a DX12 application, focus will switch from the launched program to CodeXL, and you will see a central preview image of the frames being rendered.

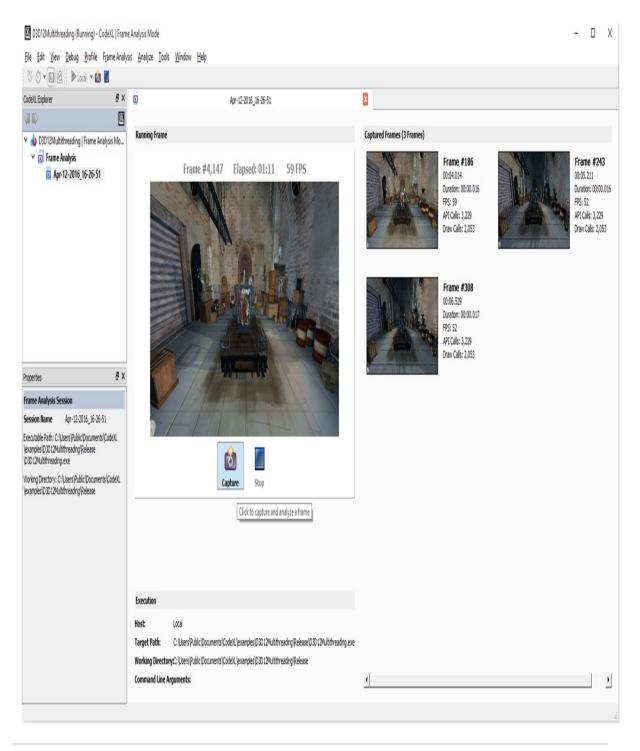




Help > Using CodeXL > Frame Analysis > Capturing Frames

Capturing Frames

- Once a frame capture session is running, press the Capture button either underneath the central preview image or in the toolbar to capture a frame. Once a frame capture has been initiated, it may take a few seconds to complete. The capture is complete when the capture button returns to an enabled state and a small preview image of the captured frame appears in the margin on the right hand side of the window.
- 2) Continue to capture as many frames as desired, waiting for a previous capture to finish before initiating a new one. The right hand margin will provide a running count of the captured frames, as well as a small preview image and basic data for each.
- 3) When stopping the capture session, press the Stop button either underneath the central preview image or in the toolbar.



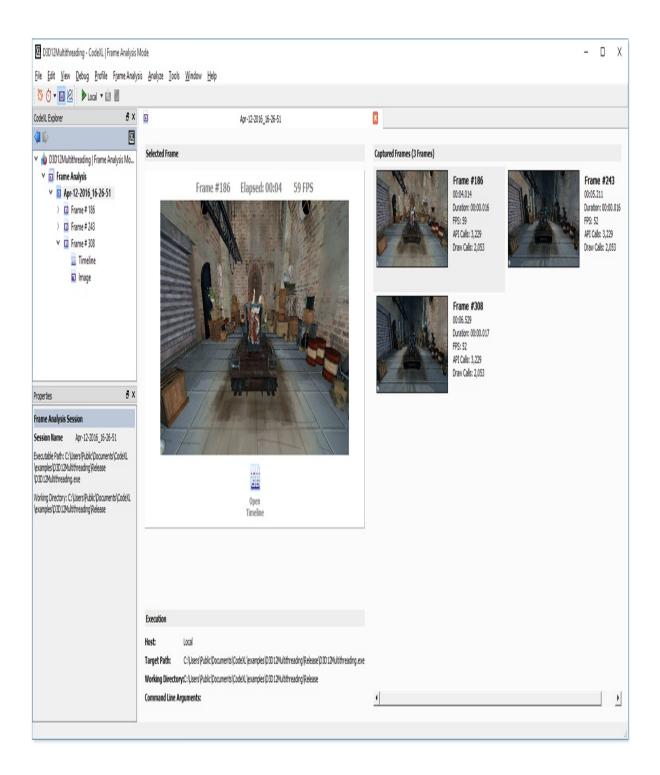




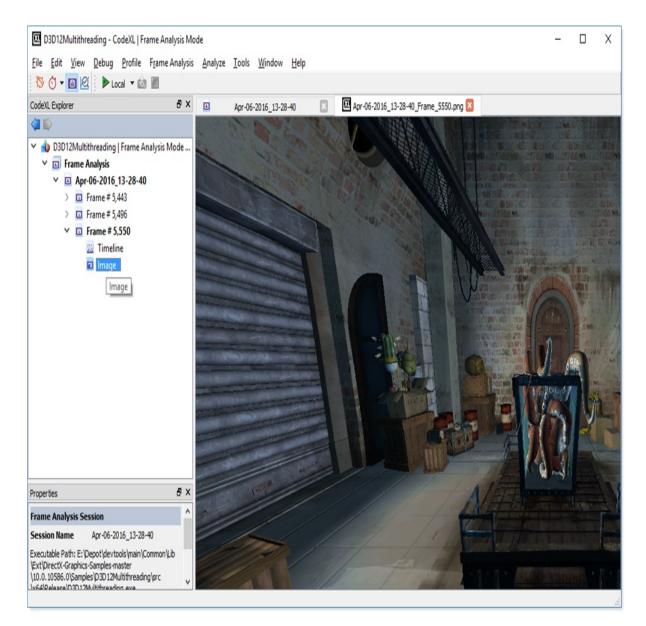
Help > Using CodeXL > Frame Analysis > Viewing Capture Data

Viewing Capture Data

- After stopping a capture session, the captured frames will appear in the left hand pane CodeXL Explorer Tree under "(**Project name here**) | Frame Analysis Mode -> Frame Analysis".
- 2) Expand the desired capture session node (capture sessions are named with the date and time of capture).
- 3) Select the desired frame sub-node. There should be a Timeline and Image item for each frame.



4) Double click the Image item to bring up a full resolution image of the captured frame.



5) Double click the timeline item to bring up GPU and CPU trace data.

🖾 D3D12Multithreading - CodeXL (Debug	binaries) Frame Analy	sis Mode										_		Х
File Edit View Debug Profile Fra	me Analysis Analyze	Tools Wi	indow Help)										
🖏 🔿 🕶 🔟 🖉 🕨 Local 🔹 🛍 🔳	X 🖗 🗟 🥹 í				*									
odeXL Explorer (Debug binaries) 🗗 🗙		23.00	18-2016_12-34		223	(×							
X 🗐								30.918		38,883 (115		69.8	302
🖞 🔬 D3D12Multithreading Frame An	Duration	▼ 0.30	- 00											
	Top 20 GPU Cmd	o.22	25 -											
Y 🖬 May-18-2016_12-34-10	Top 20 CPU API	Calls 0.19	50 -											
 Frame # 456 Timeline 	🗌 📕 All GPU Cmds	0.07	75 -						6					
🖬 lmage	All CPU API Calls		0.000			17.609		31.218			F2 678	նդե		
> III Frame # 500		0.00	30.918	The second se		40.639	1	50.360		1	60.081			
> 🛄 Frame # 516	Duration			1	1 1	8 1	(I)	0 5 5		1	8	<u>с с т</u>	1	
> 🛄 Frame # 545														
>	Inread 7848													ļ
> 🛄 Frame # 603	Thread 13044										CPU PI	ROCESSING L	ATER	1
	Thread 13544											FRAME/S		
	Thread 13620							الما المحاجبي						L
	GPU Queues			-										i
						_								
	Queue 1 Direct		EARLIE	and the second se		_								l
	API Calls		FRAME/	5										
			۲							- 11			>	
	Call History													
	Thread 7848 Thre	ad 13044	Thread 13544	Thread	13620		Q	ueue 1 Direct						
	Inc Interfa Call	Param	neters		CPUT Re	s Start T End Ti	^ In	c CmdL Call	Parameters			GPUT Res Start	t T End Ti	
	3 ID3 GetCPU)esc			4.40 0.	34.2 34.2	1	. Cm Drawlnde	xe 3312, 1, 66567	6, 1933	19, 0	113 v 63.6	63.7	
	3 ID3 SetGrap					34.2 34.2		A CONTRACTOR OF CONTRACTOR	xe 969, 1, 682830		-	258 v 63.7	1.	
	3 ID3 SetGrap					. 34.2 34.3			xe 1920, 1, 70120			10.5 v 64.0.	-	
	3 ID3 OMSetR							1200 C 10	xe 48, 1, 46971, 1		OWSON 1	9.03 v 67.8		
	3 ID3 BeginEv 3 ID3 GetGPU		rker drawing :	scene pa		. 34.3 34.3 . 34.3 34.3			xe 7809, 1, 70696 xe 7809, 1, 73026			10.2 v 64.0 5.03 v 64.0		
		/ESC			J.00 U.	, 34,3m 34,3m	¥ []	. cm., Diawinue	xe 7009, 1, 73020	0, 2122	94, 0	JiUJiii Viii 04iU	04.0	
perties 🗗 🗙	Hotspot Summary	in an			1000 10	_								
ame Analysis Session	CPU API Summary	GPU Comman	ids Summary	Comman	d Lists Summ	ary								
ssion Name May-18-2016_12-34-10	9 Command Lists	. v	o		4 1	1 (1)	102.10	meline selection scop				List7 commar	nds	_
cutable Path: C:\Users\Public\Documents	Command List Exec		internet and internet in the	Contraction of the local division of the		and a second second second second	Handle			1		Time 258.666 µs		
	CmdList7 4.97. CmdList6 3.32	a sector and a sector and a sector a s	<u>59.283 ms</u> <u>6</u> 64.262 ms <u>6</u>			AND ADDRESS OF THE OWNER	des spinotopie	009597F2CD20 009597F2BA90		2	1488 1487	<u>208.000 µs</u> 113.926 µs		
odeXL\examples\D3D12Multithreading		104						009597F2CB10		3	1037	76.000 µs		
odeXL\examples\D3D12Multithreading elease\D3D12Multithreading.exe		ms	34.848 ms 🛛 3	1.332 III2	745							10,000 µ3		
eculate rain: C: (bears public poculients ode)/L [examples [D3012Multithreading elease [D3012Multithreading.exe orking Directory: C: (Users [Public]Documents ode)/L [examples [D3012Multithreading elease	and the second second		<u>34.848 ms</u> <u>3</u> 67.589 ms <u>7</u>	70.569 ms		Queue 1 Dire	ct 0:0000			4	1287	42.963 µs		

The timeline view displays the collected frame data. See:

- <u>The Frame Timeline View</u>
- Navigating the Frame Timeline View

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Frame Analysis > Viewing Capture Data > The Frame Timeline View

The Frame Timeline View

🛛 D3D12Multithreading - CodeXL (Debu	g binaries) Frame Analysis Mode							- [X
File Edit View Debug Profile Fr	ame Analysis Analyze Tools	Window Help							
🖔 🔿 🕶 🔟 🕨 🕨 🖬			\$ h						
CodeXL Explorer (Debug binaries) 🗗 🗙		y-18-2016_12-34-10 - Frame	253	×					
						0.588 ms			
🖌 👜 D3D12Multithreading Frame An	Duration 🔻 C	0.300							
Y Frame Analysis	Top 20 GPU Cmds	0.225 -							
✓ ■ May-18-2016_12-34-10		0.150 -							
🗙 🔟 Frame # 456	serve and an entertained								I.
🚟 Timeline	All GPU Cmds	0.000	1 17.60	19 1	35.218		52,8781		
🖬 Image	All CPU API Calls	0.000	<u>, </u>				\$2,878	<u> </u>	
>		42.027	42.17	14	42.321		42.468	<u>e e a</u>	-
 III Frame # 516 III Frame # 545 	Duration	2							
> 🔟 Frame # 561	🖯 Timeline								
> 🔟 Frame # 603	CPU Threads (4 of 4)	3			1111111				^
	Thread 7848		ec	e		8	ł	ec fi	T
	Thread 13044					ec	1	í í	ec
	Thread 13544	1 1 1		otDa	sRootDescr xed		1	exed	
	Thread 13620								
				IIIII					
	GPU Queues								
	□ Queue 1 Direct	<							×
						-			/
	Call History	4			5				
	Thread 7848 Thread 13044		13620		Queue 1 Direct				
	Inc Interfa Call Par	rameters	CPUT Res Sta	rt T End Ti 🔥	Inc CmdL Call	Parameters		GPUT Res Start T	End Ti
					a second s				
	3 ID3 GetCPUDesc		4.40 0 34.		1 Cm Drawindexe.			113 v 63.6	
	3 ID3 SetGraphics 2, 0		5.86 v 34.	2 34.2	1 Cm DrawIndexe.	. 969, 1, 682830, 1	97161, 0	258 v 63.7	64.0
	3 ID3 SetGraphics 2, C 3 ID3 SetGraphics 1, C	0x0000000100260A20	5.86 v 34. 17.1 v 34.	2 34.2 2 34.3	1 Cm DrawIndexe. 1 Cm DrawIndexe.	969, 1, 682830, 1 1920, 1, 701205,	97161, 0 204469, 0	258 v 63.7 (10.5 v 64.0 (64.0 64.0
	3 ID3 SetGraphics 2, 0 3 ID3 SetGraphics 1, 0 3 ID3 OMSetRend 1, 0	0x0000000100260A20 0x000000959F5BF780, FAL	5.86 v 34. 17.1 v 34. 7.33 v 34.	2 34.2 2 34.3 3 34.3	1 Cm DrawIndexe. 1 Cm DrawIndexe. 1 Cm DrawIndexe.	969, 1, 682830, 1 1920, 1, 701205, 48, 1, 46971, 144	97161, 0 204469, 0 49, 0	258 v 63.7) 10.5 v 64.0) 9.03 v 67.8)	64.0 64.0 67.8
	3 ID3 SetGraphics 2, 0 3 ID3 SetGraphics 1, 0 3 ID3 OMSetRend 1, 0 3 ID3 BeginEvent 0, V	0x0000000100260A20 0x000000959F5BF780, FAL	5.86 v 34. 17.1 v 34. 7.33 v 34. 2.93 v 34.	2 34.2 2 34.3 3 34.3 3 34.3	1 Cm DrawIndexe.	 969, 1, 682830, 1 1920, 1, 701205, 48, 1, 46971, 144 7809, 1, 706965, 	97161, 0 204469, 0 49, 0 206464, 0	258 v 63.7 (10.5 v 64.0 (9.03 v 67.8 (10.2 v 64.0 (64.0 64.0 67.8 64.0
	3 ID3 SetGraphics 2, 0 3 ID3 SetGraphics 1, 0 3 ID3 OMSetRend 1, 0	0x0000000100260A20 0x000000959F5BF780, FAL	5.86 v 34. 17.1 v 34. 7.33 v 34.	2 34.2 2 34.3 3 34.3 3 34.3	1 Cm DrawIndexe. 1 Cm DrawIndexe. 1 Cm DrawIndexe.	 969, 1, 682830, 1 1920, 1, 701205, 48, 1, 46971, 144 7809, 1, 706965, 	97161, 0 204469, 0 49, 0 206464, 0	258 v 63.7) 10.5 v 64.0) 9.03 v 67.8)	64.0 64.0 67.8 64.0
roperties 5 X	3 ID3 SetGraphics 2, 0 3 ID3 SetGraphics 1, 0 3 ID3 OMSetRend 1, 0 3 ID3 OMSetRend 1, 0 3 ID3 GetGPUDesc 0, V 3 ID3 GetGPUDesc 0	0x0000000100260A20 0x000000959F5BF780, FAL	5.86 v 34. 17.1 v 34. 7.33 v 34. 2.93 v 34.	2 34.2 2 34.3 3 34.3 3 34.3	1 Cm DrawIndexe.	 969, 1, 682830, 1 1920, 1, 701205, 48, 1, 46971, 144 7809, 1, 706965, 	97161, 0 204469, 0 49, 0 206464, 0 212294, 0	258 v 63.7 (10.5 v 64.0 (9.03 v 67.8 (10.2 v 64.0 (64.0 64.0 67.8 64.0
	3 ID3 SetGraphics 2, 0 3 ID3 SetGraphics 1, 0 3 ID3 OMSetRend 1, 0 3 ID3 OMSetRend 1, 0 3 ID3 GetGPUDesc 0, V 3 ID3 GetGPUDesc 0 CPU API Summary GPU Comm GPU Comm	0000000100260A20 000000959F5BF780, FAL Worker drawing scene pa 6	5.86 v 34. 17.1 v 34. 7.33 v 34. 2.93 v 34.	2 34.2 2 34.3 3 34.3 3 34.3	1 Cm DrawIndexe.	 969, 1, 682830, 1 1920, 1, 701205, 48, 1, 46971, 144 7809, 1, 706965, 	97161, 0 204469, 0 49, 0 206464, 0 212294, 0 7	258 v 63.7 1 10.5 v 64.0 1 9.03 v 67.8 1 10.2 v 64.0 1 5.03 v 64.0 1	64.0 64.0 67.8 64.0
ame Analysis Session	3 ID3 SetGraphics 2, 0 3 ID3 SetGraphics 1, 0 3 ID3 OMSetRend 1, 0 3 ID3 OMSetRend 1, 0 3 ID3 GetGPUDesc 0, V 3 ID3 GetGPUDesc 0	0000000100260A20 000000959F5BF780, FAL Worker drawing scene pa	5.86 v 34. 17.1 v 34. 7.33 v 34. 2.93 v 34. 5.86 0 34.	2 34.2 2 34.3 3 34.3 3 34.3	1 Cm DrawIndexe.	 969, 1, 682830, 1 1920, 1, 701205, 48, 1, 46971, 144 7809, 1, 706965, 7809, 1, 730263, 	97161, 0 204469, 0 49, 0 206464, 0 212294, 0 7	258 v 63.7 (10.5 v 64.0 (9.03 v 67.8 (10.2 v 64.0 (64.0 64.0 67.8 64.0
rame Analysis Session ession Name May-18-2016_12-34-10	3 ID3 SetGraphics 2, 0 3 ID3 SetGraphics 1, 0 3 ID3 OMSetRend 1, 0 3 ID3 OMSetRend 1, 0 3 ID3 GetGPUDesc 0, V 3 ID3 GetGPUDesc 0 CPU API Summary GPU Comm GPU Comm 9 Command Lists Command List Execution time	0x000000100260A20 0x00000959F5BF780, FAL Worker drawing scene pa Morker drawing scene pa G Commani Start time End time	5.86 v 34. 17.1 v 34. 7.33 v 34. 2.93 v 34. 5.86 0 34. d Lists Summary # of commands	2 34.2 2 34.3 3 34.3 3 34.3 3 34.3 GPU queue	1 Cm DrawIndexe. Use timeline selection scope Handle	 969, 1, 682830, 1 1920, 1, 701205, 48, 1, 46971, 144 7809, 1, 706965, 	97161, 0 204469, 0 49, 0 206464, 0 212294, 0 7 Top 20 Cmd	258 v 63.7 (10.5 v 64.0 (9.03 v 67.8 (10.2 v 64.0 (5.03 v 64.0 (List7 commands Time	64.0 64.0 67.8 64.0
ame Analysis Session ession Name May-18-2016_12-34-10 recutable Path: C: (Users)Public/Documents odeXL (examples)D3D12Multithreading	3 ID3 SetGraphics 2, 0 3 ID3 SetGraphics 1, 0 3 ID3 SetGraphics 1, 0 3 ID3 OMSetRend 1, 0 3 ID3 BeginEvent 0, V 3 ID3 GetGPUDesc 9 Hotspot Summary GPU Comm 9 Command Lists Command List ComdList7 4.972 ms 4.972 ms	5x000000100260A20 5x000000959F5BF780, FAL Worker drawing scene pa Annode Summary Start time 59.283 ms 64.255 ms	5.86 v 34. 17.1 v 34. 7.33 v 34. 2.93 v 34. 5.86 0 34. d Lists Summary # of commands 342	2 34.2 2 34.3 3 34.3 3 34.3 ↓ 3 34.3 ↓ GPU queue Queue 1 Direct	1 Cm DrawIndexe. Use timeline selection scope Handle 0x0000009597F2CD20 Mathematical Science Sci	 969, 1, 682830, 1 1920, 1, 701205, 48, 1, 46971, 144 7809, 1, 706965, 7809, 1, 730263, 	97161, 0 204469, 0 49, 0 205464, 0 212294, 0 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7	258 v 63.7 (10.5 v 64.0 (9.03 v 67.8 (10.2 v 64.0 (5.03 v 64.0 (ILISET commands Time <u>258.666 us</u>	64.0 64.0 67.8 64.0 64.0
ame Analysis Session ession Name May-18-2016_12-34-10 ecutable Path: C:\Users\Public\Documents odeXL\examples\D3D12Multithreading elease\D3D12Multithreading.exe	3 ID3 SetGraphics 2, 0 3 ID3 SetGraphics 1, 0 3 ID3 OMSetRend 1, 0 3 ID3 OMSetRend 1, 0 3 ID3 BeginEvent 0, V 3 ID3 GetGPUDesc 0 CPU API Summary GPU Comm 9 Command Lists Command Lists ComdList7 4.972 ms CmdList6 CmdList6 3.320 ms 3.320 ms 0	0x000000100260A20 0x000000959F5BF780, FAL Norker drawing scene pa 6 commands Start time End time 59.283 ms 64.262 ms 67.582 ms	5.86 v 34. 17.1 v 34. 7.33 v 34. 2.93 v 34. 5.86 0 34. d Lists Summary # of commands 342 342	2 34.2 2 34.3 3 34.3 3 34.3 3 34.3 GPU queue Queue 1 Direct Queue 1 Direct	Lue timeline selection scope Handle 0x0000009597F2CD20 0x000009597F2BA90	 969, 1, 682830, 1 1920, 1, 701205, 48, 1, 46971, 144 7809, 1, 706965, 7809, 1, 730263, 	97161, 0 204469, 0 49, 0 206464, 0 212294, 0 7 Top 20 Cmd ≢ index 1 1488 2 1487	258 v 63.7 0 10.5 v 64.0 0 9.03 v 67.8 0 10.2 v 64.0 0 5.03 v 64.0 0 ILISET COMMANDES Time 258.666 µs 113.926 µs	64.0 64.0 67.8 64.0 64.0
rame Analysis Session ession Name May-18-2016_12-34-10 xecutable Path: C:{Users\Public\Documents IodeXL\examples\D3D12Multithreading telease\D3D12Multithreading.exe forking Directory: C:{Users\Public\Documents	3 ID3 SetGraphics 2, 0 3 ID3 SetGraphics 1, 0 3 ID3 SetGraphics 1, 0 3 ID3 OMSetRend 1, 0 3 ID3 BeginEvent 0, V 3 ID3 GetGPUDesc 0 4 Hotspot Summary GPU Comm 9 Command List Ecmulaist Command List CmdList7 4.972 ms CmdList6 CmdList6 3.320 ms SetGraphics	6 Source 6 Commany Commany Start time End time 59,283 ms 64,262 ms 64,262 ms 37,593 ms	5.86 v 34. 17.1 v 34. 7.33 v 34. 2.93 v 34. 5.86 0 34. d Lists Summary # of commands 342 342	2 34.2 2 34.3 3 34.3 3 34.3 3 34.3 GPU queue Queue 1 Direct Queue 1 Direct Queue 1 Direct	1 Cm DrawIndexe.	 969, 1, 682830, 1 1920, 1, 701205, 48, 1, 46971, 144 7809, 1, 706965, 7809, 1, 730263, 	97161, 0 204469, 0 49, 0 206464, 0 212294, 0 7 Top 20 Cmd # index 1 1488 2 1487 3 1037	258 v 63.7 0 10.5 v 64.0 0 9.03 v 67.8 0 10.2 v 64.0 0 5.03 v 64.0 0 Itist7 commands Time <u>258.666 us</u> <u>113.926 us</u> <u>76.000 us</u>	64.0 64.0 67.8 64.0
roperties 5 × rame Analysis Session ession Name May-18-2016_12-34-10 xecutable Path: C: (Users) Public (Documents CodeXL (examples (D3D 12Multithreading Release (D3D 12Multithreading, exe /orking Directory: C: (Users) Public (Documents CodeXL (examples (D3D 12Multithreading Release	3 ID3 SetGraphics 2, 0 3 ID3 SetGraphics 1, 0 3 ID3 OMSetRend 1, 0 3 ID3 OMSetRend 1, 0 3 ID3 BeginEvent 0, V 3 ID3 GetGPUDesc 0 CPU API Summary GPU Comm 9 Command Lists Command Lists ComdList7 4.972 ms CmdList6 CmdList6 3.320 ms 3.320 ms 0	0x000000100260A20 0x000000959F5BF780, FAL Norker drawing scene pa 6 commands Start time End time 59.283 ms 64.262 ms 67.582 ms	5.86 v 34. 17.1 v 34. 7.33 v 34. 2.93 v 34. 5.86 0 34. d Lists Summary # of commands 342 342 342 341	2 34.2 2 34.3 3 34.3 3 34.3 3 34.3 4 3 34.3 4 9 9 9 9 9 9.	1 Cm DrawIndexe.	 969, 1, 682830, 1 1920, 1, 701205, 48, 1, 46971, 144 7809, 1, 706965, 7809, 1, 730263, 	97161, 0 204469, 0 49, 0 206464, 0 212294, 0 7 Top 20 Cmd # index 1 1488 2 1487 3 1037 4 1287	258 v 63.7 0 10.5 v 64.0 0 9.03 v 67.8 0 10.2 v 64.0 0 5.03 v 64.0 0 ILISET COMMANDES Time 258.666 µs 113.926 µs	64.0 64.0 67.8 64.0

The frame timeline view contains the following elements (from bottom to top). Each of the elements' numbers is in the screenshot above.

1. A navigation chart that allows you to zoom in and out, view the whole frame timeline or focus on a fragment of it, and display API calls duration, count and concurrency.

2. A collapsed detailed view of the API calls and their durations/count/concurrency.

3. A timeline chart visualizing the API calls over the frame timeline.

4. An API calls table for each of the CPU threads.

5. A commands table for each of the GPU command queues.

6. Summary tables that summarize the time consumption for each of the API types in the frame.

7. A Top 20 calls table for the currently selected API in the summary table.

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Frame Analysis > Viewing Capture Data > View and analyze data in API Summary tables

View and analyze data in API Summary tables

The bottom pane of the trace view contain hotspot Summary tables. The table are summarizing the API call, and display each of the API types significant times. These tables can be used for quick identification of performance bottlenecks in the analyzed frame.

The table below specified each of the API table columns.

CPU API Summary GPU Commands Summary		Command Lists Summary									
9 Command Lists				49 3	[Use timeline selection scope		Top 20	Cmd	List7 commands	
Command List	Execution time	Start time	End time	# of commands	GPU queue	Handle	^	#	index	Time	
CmdList7	4.972 ms	<u>59.283 ms</u>	<u>64.255 ms</u>	342	Queue 1 Direct	0x0000009597F2CD20		1	1488	258.666 µs	
CmdList6	3.320 ms	<u>64.262 ms</u>	<u>67.582 ms</u>	342	Queue 1 Direct	0x0000009597F2BA90		2	1487	<u>113.926 µs</u>	
CmdList1	3.145 ms	<u>34.848 ms</u>	<u>37.993 ms</u>	342	Queue 1 Direct	0x0000009597F2CB10		3	1037	<u>76.000 µs</u>	
	2.980 ms	<u>67.589 ms</u>	70.569 ms	341	Queue 1 Direct			4	1287	42.963 µs	
CmdList3	2.223 ms	38.001 ms	40.223 ms	342	Queue 1 Direct	0x0000009597F2CDD0	~	5	1139	30.074 µs	

CPU \ GPU properties:

Name	
Call	The API call name
Max Time	The maximum duration for this call type. Clicki timeline.
Min Time	The minimum duration for this call type. Clicki: timeline.
Average Time	The average duration for this API type.
Cumulative Time	The cumulative duration for all calls with this A
% of total	The percentage of all this API type calls, from t

Time	
# of Calls	The calls count for this API type

CodeXL 2.1 supports both DX12 and Vulkan API. The hotspot summary table shows DX12 Command list \ Vulkan command buffer properties:

Name	
Command List\Buffer	The command list \ buffer name
Execution Time	The duration for this command.
Start Time	The start time of this command list \ buffer. Cli
End Time	The end time of this command list \ buffer. Clic
# of commands	The number of commands in this command list
GPU Queue	The GPU queue associated with this command
Handle	The address of this command list \ buffer.

 Use timeline selection scope: when checked, the summary tables will reflect the selected timeline frame (which is painted within the red boundaries above). When un-checked, the summary tables will display the whole frame time range.

Selection of a line in the API summary table, will fill the top 20 calls, sorted by the time.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Frame Analysis > Viewing Capture Data > View data in the Summary Top 20 Table

View data in the Summary Top 20 Table

The screenshot below shows the API calls table, with "DrawIndexedInstanced" selected. The Top 20 table will display the top 20 time consuming calls to DrawIndexedInstanced.

CPU API Summary	GPU Commands	Summary (Command List	s Summary									
Call History Aggregate	d Data	Ċ.			Use timeli	ne selection scope			Top 2	0 DrawInd	dexed	instanced calls	(CPU)
Call		Max Time	Min Time	Average Time	Cumulative Time	% of total Time	# of Calls	۸	#	Thread Id	index	Time	^
DrawIndexedInstan	ced	<u>124.666 µs</u>	<u>2.444 µs</u>	5.515 µs	11.306 ms	61.057%	2050		1	13044	672	124.666 µs	
SetGraphicsRootDes	scriptorTable	<u>120.267 µs</u>	<u>1.956 µs</u>	5.734 µs	5.999 ms	32.395%	1046		2	7848	984	48.400 µs	
RSSetViewports		<u>64.044 µs</u>	<u>4.400 µs</u>	20.696 µs	124.177 µs	0.671%	6		3	13544	625	<u>48.400 µs</u>	
ExecuteCommandL	ists	<u>63.067 µs</u>	<u>46.934 µs</u>	55.000 µs	110.001 µs	0.594%	2		4	13544	531	40.088 µs	
OMSetRenderTarge	ts	40.578 µs	5.867 µs	14.829 µs	88.978 µs	0.481%	6	~	5	13544	129	38.622 µs	

Name	
#	The number of the call (1 for the top time const
Thread Id	The thread on which the call was executed
Index	The index within the thread
Time	The call time in ms. Clicking on the link in this





Help > Using CodeXL > Frame Analysis > Viewing Capture Data > View data in CPU API Tables

View data in CPU API Tables

The CPU and GPU API table monitor each of the frames API calls. The table below specified each of the API table columns.

API Calls

Thread	7804 Thread 16332 T	hread 16948 Thread 2077	72					
Index	Interface	Call	Parameters	CPU Time	Result	Start Time	End Time	^
10	ID3D12GraphicsComm	OMSetStencilRef	0	7.333 µs	void	4.536 ms	4.543 ms	
11	ID3D12GraphicsComm	SetGraphicsRootDescri	2, 0x000000020165000	2.445 µs	void	4.546 ms	4.549 ms	
12	ID3D12GraphicsComm	SetGraphicsRootDescri	1, 0x00000000201659A0	3.423 µs	void	4.551 ms	4.554 ms	
13	ID3D12GraphicsComm	OMSetRenderTargets	0, 0x000000000000000, FALSE, 0x000000D7D3BD	3.911 µs	void	4.557 ms	4.561 ms	
14	ID3D12DescriptorHeap	GetGPUDescriptorHand		977 ns	0x000000020	4.563 ms	4.564 ms	
15	ID3D12GraphicsComm	SetGranhicsRootDescri	0, 0x0000000020165000	1.956 µs	void	4.568 ms	4.570 ms	
16	ID3D12GraphicsComm	D3D12DescriptorHeap	0, Worker drawing shadow pass, 60	2.934 µs	void	4.573 ms	4.576 ms	
17	ID3D12GraphicsComm	DrawIndexedInstanced	15198, 1, 0, 0, 0	4.400 µs	void	5.223 ms	5.228 ms	
18	ID3D12GraphicsComm	DrawIndexedInstanced	300, 1, 15936, 6225, 0	978 ns	void	5.235 ms	5.236 ms	
19	ID3D12GraphicsComm	DrawIndexedInstanced	438, 1, 16974, 6460, 0	978 ns	void	5.239 ms	5.240 ms	

Name	
Index	The API call index for this thread.
Interface	The API interface.
Call	The API call name
Parameters	A string describing the parameters for this call
CPU Time	The duration of this API call on the CPU
Result	The API return value
Start Time	The time (from the frame start time in ms) the
End Time	The time (from the frame start time in ms) the

* Notice: BeginEvent, EndEvent, and SetMarker API calls are

marked in yellow for easy navigation.





Help > Using CodeXL > Frame Analysis > Viewing Capture Data > View data in GPU API Tables

View data in GPU API Tables

The CPU and GPU API table monitor each of the frames API calls. The table below specified each of the API table columns.

Queue	1 Direct							
Index	CmdList #	Call	Parameters	GPU Time	Result	Start Time	End Time	^
1487	CmdList7	DrawIndexedInstanced	3312, 1, 665676, 193319, 0	113.926 µs	void	63.633 ms	63.747 ms	
1488	CmdList7	DrawIndexedInstanced	969, 1, 682830, 197161, 0	258.666 µs	void	63.750 ms	64.009 ms	
1489	CmdList7	DrawIndexedInstanced	1920, 1, 701205, 204469, 0	10.519 µs	void	64.045 ms	64.056 ms	
1490	CmdList8	DrawIndexedInstanced	48, 1, 46971, 14449, 0	9.037 µs	void	67.873 ms	67.882 ms	
1491	CmdList7	DrawIndexedInstanced	7809, 1, 706965, 206464, 0	10.222 µs	void	64.059 ms	64.069 ms	
1492	CmdList7	DrawIndexedInstanced	7809. 1. 730263. 212294. 0	5.037 us	void	64.088 ms	64.093 ms	V

Name			
Index	The API call index for this thread.		
Cmd List #	The index of the command list \ buffer.		
Call	The API call name.		
Parameters	A string describing the parameters for this call		
GPU Time	The duration of this API call on the GPU		
Result	The API return value		
Start Time	The time (from the frame start time in ms) the		
End Time	The time (from the frame start time in ms) the		





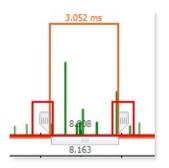
Help > Using CodeXL > Frame Analysis > Viewing Capture Data > Navigating the Frame Timeline View

Navigating the Frame Timeline View

A graphics frame can be very busy and contains tens of thousands of API calls. The following buttons and UI elements are useful in navigating the frame timeline and highlighting API calls of interest:

Focused Timeline Fragment

Use the left and right handles to expand/reduce the focused timeline fragment in and out:



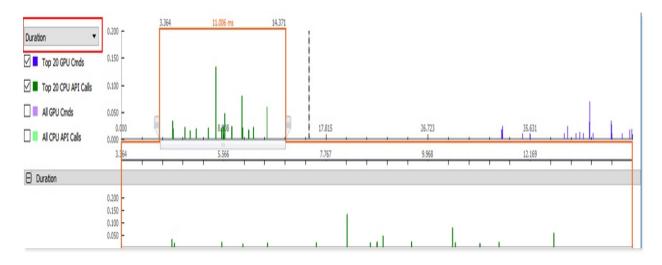
Navigation Bar Visualizations

The navigation bar drop-list contains 3 visualization aids:

- Duration
- · Count
- · Concurrency

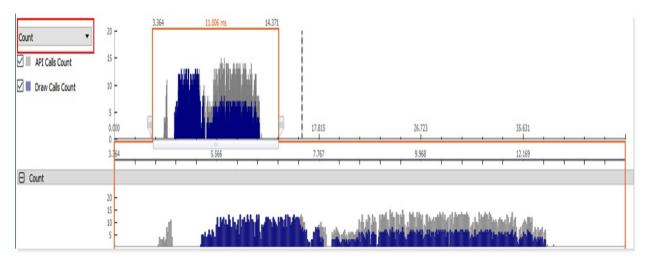
API Calls Duration

Select "Duration" in the top left combo box, to view the duration of API calls:



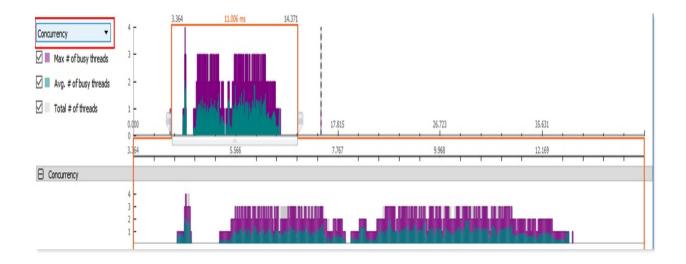
API Calls Count

Select "Count" in the top left combo box, to view the API calls count for each time fragment:



API Calls Concurrency

Select "Concurrency" in the top left combo box, to view the max / average busy threads concurrency over the frame timeline:



Timeline chart and API Calls tables

Double click a CPU API table item, and the timeline chart will zoom to the corresponding timeline item and highlight it. If there is a linked GPU API item, it will also be highlighted in the GPU API table

🖯 Timeline												
111Cau 70 10												1
Thread 13044						CPU PR	OCESSING LATER FF	RAME/S				1
Thread 13544												1 1
Thread 13620												1
- GPU Queues												Ī
⊡Queue 1Direct												
API Calls				aw and a second	IndexedIr	tano Dra	wIndexedInstanced		e e e e e e e e xedie		ell elekter	
								() () () () () () () () () ()				-
Command Lists									CmdList6			۷
Command Lists	<			CmdList	1				CmdList6		>	¥
Cormand Lists	ζ			CmdList	7				CmdList6		>	v
Call History	read 13544 Thread 13620			Cindust		Queue 1 Direct			CmdList6		>	
Call History Thread 7848 Thread 13044 Thr	read 13544 Thread 13620 Parameters	CPU Tim Res	ι Start Tim			Queue 1 Direct ndex CmdList #	Call	Parameters	CmdList6	Result Start Time	> End Time	v
 Call History Thread 7848 Thread 13044 Thr Inde Interface Call 		CPU Tim Res 4.400 µs v		End Tim	(Call DrawIndexedInstanced	Parameters 168, 1, 659319, 191765, 0	GPU Time	Result Start Time void 63.620 ms	1.000.00000000	v
Cal History Thread 7848 Thread 13044 Th Inde Interface Call F 10 ID3D1 DrawIndexedInst 1	Parameters		57.339	End Tim 57.344	(^ [1	ndex CmdList #			GPU Time		63.623 ms	Υ.
Call History Thread 7848 Thread 13044 Inde Interface Call 10 ID3D1 DrawIndexedInstr 1	Parameters 1617, 1, 662721, 192605, 0	4.400 µs v	57.339 57.376	End Tim 57.344 57.379	(^ 1	ndex CmdList# 484 CmdList7	DrawIndexedInstanced	168, 1, 659319, 191765, 0	GPU Time 3.259 µs 9.481 µs	void 63.620 ms	63.623 ms 64.354 ms	
Call History Thread 7848 Thread 13044 Thread 13044 Inde Interface Call F 10 ID3D1 DrawIndexedInst 1 10 ID3D1 SetGraphicsRoot 0 10 ID3D1 DrawIndexedInst 3	Parameters 1617, 1, 662721, 192605, 0 0, 0x0000000100260720	4.400 μs v 3.423 μs v	57.339 57.376 57.450	End Tim 57.344 57.379 57.454	(1 1 1	ndex CmdList# 484 CmdList7 485 CmdList6	DrawindexedInstanced DrawindexedInstanced	168, 1, 659319, 191765, 0 438, 1, 34620, 10577, 0	GPU Time 3.259 µs 9.481 µs 3.852 µs	void 63.620 ms void 64.345 ms	63.623 ms 64.354 ms 63.630 ms	
Call History Thread 7848 Thread 13044 Th Inde Interface Call Inu. ID3D1 DrawIndexedInst I I ID3D1 DrawIndexedInst I I ID3D1 DrawIndexedInst I III. ID3D1 DrawIndexedInst I III. ID3D1 ID3	Parameters 1617, 1, 662721, 192605, 0 0, 0x000000100260720 3312, 1, 665676, 193319, 0 0, 0x0000001002607C0	4.400 µs v 3.423 µs v 3.911 µs v	57.339 57.376 57.450 57.490	End Tim 57.344 57.379 57.454 57.493	(484 CmdList# 484 CmdList7 485 CmdList6 486 CmdList7	DrawIndexedInstanced DrawIndexedInstanced DrawIndexedInstanced	168, 1, 659319, 191765, 0 438, 1, 34620, 10577, 0 1617, 1, 662721, 192605, 0	GPU Time 3.259 µs 9.481 µs 3.852 µs 113.926 µs	void 63.620 ms void 64.345 ms void 63.626 ms	63.623 ms 64.354 ms 63.630 ms 63.747 ms	Ŷ

Hotspot Summary Tables

The bottom ribbon of the timeline view contains the Hotspot Summary tables. These tables display aggregated data for API calls and command lists \ buffers, showing statistics for each type of API and the top individual calls to that API type.

CPU API Summary G	PU Command	ls Summary	Command Lists	Summary						
GPU Cmds Aggregated Da	ta					Use timeline selection scope	To	p <mark>20 Dr</mark> a	wIndexedInstanced calls (GPU)	
Call	Max Time	Min Time	Average Time	Cumulative Time	% of total Time	# of Calls	#	index	Time	
DrawIndexedInstanced	<u>258.666 µs</u>	<u>2.963 µs</u>	5.894 µs	12.084 ms	99.088%	2050	1	1488	258.666 µs	
ResourceBarrier	<u>68.000 µs</u>	<u>888 ns</u>	20.481 µs	81.925 µs	0.672%	4	2	1487	<u>113.926 µs</u>	
ClearDepthStencilView	<u>11.704 µs</u>	<u>9.777 µs</u>	10.740 µs	21.481 µs	0.176%	2	3	1502	<u>110.370 µs</u>	
ClearRenderTargetView	<u>7.852 µs</u>	7.852 µs	7.852 µs	7.852 µs	0.064%	1	4	1720	<u>90.222 µs</u>	
							5	1037	76.000 µs	

The Max Time and Min Time columns display the execution

time of the longest and shortest API call of the selected API type. These are also direct links – clicking them causes the API tables and timeline chart to display the individual call item.

The 'Top 20' table is automatically populated with the 20 longest calls of the API type selected in the API summary table.

Performance tip: The longest GPU command in the frame, is always the first API call in the Top 20 table when the timeline view is opened.

CPU API Summa	ary GPU Comma	inds Summary	Command Lis	ts Summary							
Command Lists						Use timeline select	tion scope	Тор	20 Cmd	List7 commands	
Command List	Execution time	Start time	End time # of	f commands G	GPU queue	Handle	^	#	index	Time	1
CmdList7	4.972 ms	<u>59.283 ms</u>	<u>64.255 ms</u> 342	Q	Jueue 1 Direct	0x000009597F2CD20		1	1488	258.666 µs	
CmdList6	3.320 ms	<u>64.262 ms</u>	<u>67.582 ms</u> 342	Q	Queue 1 Direct	0x000009597F2BA90		2	1487	<u>113.926 µs</u>	
CmdList1	3.145 ms	<u>34.848 ms</u>	<u>37.993 ms</u> 342	Q	Queue 1 Direct	x0000009597F2CB10		3	1037	<u>76.000 µs</u>	
	2.980 ms	<u>67.589 ms</u>	70.569 ms 341	Q	Queue 1 Direct			4	1287	<u>42.963 µs</u>	
CmdList3	2.223 ms	38.001 ms	40.223 ms 342	Q	Queue 1 Direct	0x000009597F2CDD0		5	1139	<u>30.074 µs</u>	

For command lists $\$ buffers the hotspot table displays the command lists $\$ buffers which were executed during the captured frame.

The Execution Time column displays the execution time of the longest and shortest command lists \ buffer. The start and end time are also direct links – clicking them causes the timeline chart to display the individual command list \ buffer.

The 'Top 20' table is automatically populated with the 20 longest GPU call which belongs to the command list \ buffer which is selected in the hotspot summary table.

Performance tip: The longest GPU command in the frame, is always the first API call in the Top 20 table when the timeline view is opened. $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger

GPU Debugger

The CodeXL GPU Debugging module traces application activity that makes use of OpenCL and OpenGL to provide application behavior information necessary for finding bugs and optimizing application performance.

With CodeXL, you can look inside your OpenCL and OpenGL API usage to see the effect individual commands have on application behavior.

CodeXL also lets you debug your OpenCL kernels at runtime, inspect variable values across different work item and work groups, inspect the kernel call stack, and more.

There are different ways to use the analytic capabilities of the CodeXL GPU Debugging Module: from locating bugs through removing redundant calls and errors to performing regression tests.

Whether you want to shorten debugging time, improve application quality, or optimize application performance, CodeXL displays the information you want.

The GPU debugging controls include:

- **Toolbars**
- · <u>Views</u>
- **Dialogs**





Help > Using CodeXL > GPU Debugger > GPU Debugging Toolbars

GPU Debugging Toolbars

There are two types of GPU debugging toolbars:

- Images and Buffers
- Current Work Item

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger > GPU Debugging Toolbars > Images and Buffers Toolbar

Images and Buffers Toolbar



The Images and Buffers toolbar provides control over the CodeXL **<u>object views</u>**.

	Select / pan	Change the mouse click operation from selecting pixels to panning the image view.
• • 🕎 🔛	Zoom controls	Control the zoom level of the image view.
5 0	Rotation controls	Rotate the image and data views clockwise or counterclockwise.
868 0	Channel Selection controls	For multi-channeled images and textures, select which of the RGBA channels to show in the image and data views.
	Invert	Invert the image view's displayed colors.
8))	Grayscale	Desaturate the image view's displayed image.

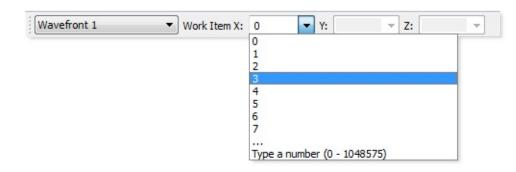
© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger > GPU Debugging Toolbars > Current Work Item toolbar

Current Work Item toolbar



The Current Work Item toolbar is shown during debugging. Use this toolbar to select the active thread.

When N-Dimensional kernel debugging is taking place (clEnqueueNDRangeKernel), use this toolbar to select the current work item.

- When looking at variables (watch view, locals view, hovering over the variable name), the values shown will be those relevant to this work item.
- When stepping through code, code locations where this work item is not valid will be skipped.
- For coordinates that have more than 8 work items, the last item in the drop down list will show the range of available work items (in the above example, there are 1048576 work items, with indexes ranging between 0 and 1048575).
- To reach a work item whose index is larger than 7, you need to type the index manually and press enter (or switch to another combo box). For example, if you want the X coordinate to be set on the index is 8192, you should type the string 8192 in the X combo box and press enter:

Wavefront 1 🔹 W	Work Item X: 8192 -	Y: 🔻 Z: 💌
-----------------	---------------------	-----------

Thread / Wavefront Selection Combo-box

This combo-box contains a list of the host threads currently active in the debugged application. Selecting a thread shows its call stack in the Call Stack view.

During kernel debugging, the active wavefronts are also shown in this combo-box. Selecting a wavefront displays its kernel source call stack.

Note that when debugging with breakpoint emulation, all workitems are gathered in a single virtual wavefront, which does not represent the actual work-item distribution on the hardware.

X, Y, Z Combo-boxes

These combo-boxes are filled with valid values for the workitem coordinates (based on the global work offset and size). If the work dimension is too low, the combo-box is disabled (for example: Z in a 2-dimensional work).

 $[\]ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger > GPU Debugging Views

GPU Debugging Views

There are six GPU debugging views.

- API Function Calls History
- Memory view
- OpenCL Multi-Watch
- Visual Studio native debugging views
- **Statistics view**
- **Object views**

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger > GPU Debugging Views > API Function Calls History View

API Function Calls History View

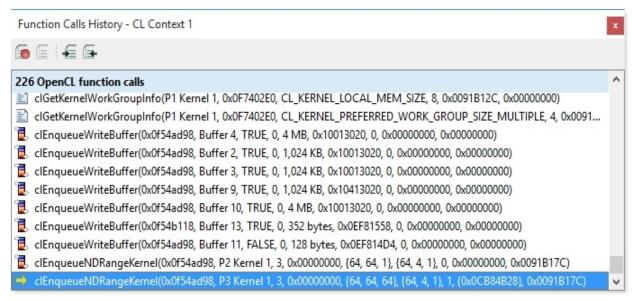
This view displays a log of OpenCL, OpenGL, OpenGL extensions, as well as WGL and glX function calls executed in each context.

Function Calls List

The function calls are displayed as a list, ordered by the time in which they were called. Each list line represents a single function call; it contains:

- the called function name,
- its arguments values, and
- the function type (as an icon).

The function types are marked as shown in the following screenshot.



The icons have the following meanings

С	OpenCL function.
—	OpenCL buffer and image function.
1	OpenCL queue function.
G	OpenGL function.
Е	OpenGL extension function.
w	A WGL function.
x	A GLX function.
	OpenGL program and shader function.
	OpenGL texture function.
臝	OpenGL buffer function.
+	GL_string_marker_GREMEDY function.

The Viewed Render Context

The list title bar shows the viewed context and the number of functions executed in it. Use the CodeXL Debugging Objects Explorer tree view to change the displayed context.

Breaked-On Function

A yellow arrow indicates the function called when the debugged process was suspended. Note that the process is suspended by CodeXL before the suspending function is executed. This lets you use the Step (F10) command to observe the effect its execution has on the API and objects.

Frame Terminators

When frame terminators are specified in the project settings, then during Frame Terminator execution the list in the context is cleared, and the function calls count is reset to 0.

Displaying Function Call Properties

Selecting a list line presents the appropriate function call in the Properties view.

HTML Log File

You can save the OpenCL / OpenGL function calls history in an HTML log file using the Record button. After pressing the Record button, a log file is created for each active context. This log file contains the details of each API function call, program and shader sources and image files for image and texture objects (if enabled in the Options dialog). To stop recording, press the record button again. To view the log file gathered so far, press the Open Current HTML Log File button.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





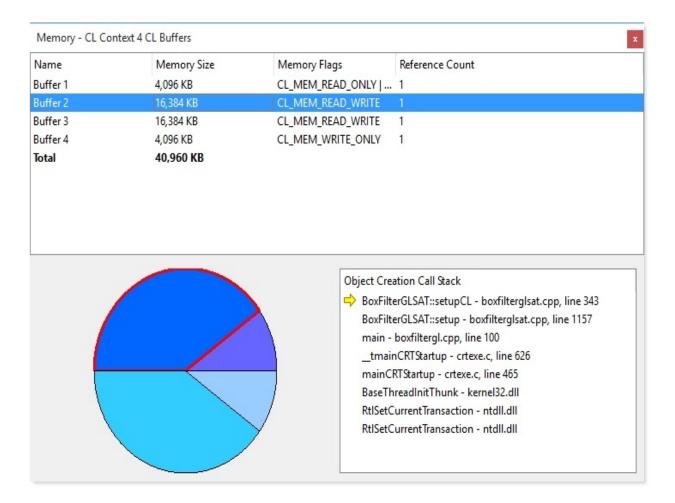
Help > Using CodeXL > GPU Debugger > GPU Debugging Views > Memory view

Memory view

This view gives you information about your compute and graphic memory consumption and usage, as well as detected graphic or compute memory leaks.

Graphic Object Details View

The Graphic Object Details view lists all the graphic memory allocated objects of the selected type in the appropriate context (for example, Context 1 textures). The columns in the list vary depending on the selected item type, displaying information relevant to the current items' memory size.



Object Creation Calls Stack View

This view displays the calls stack when the currently selected item was created. Select a stack frame to view its information. If your application has the appropriate debug information and source code, double-clicking a stack frame opens the Visual Studio Source Code editor, highlighting the appropriate line.

Graph View

This view contains details of the graphic memory consumed by the objects displayed in the Graphic Objects Details view.





Help > Using CodeXL > GPU Debugger > GPU Debugging Views > OpenCL Multi-Watch Views

OpenCL Multi-Watch Views

This view lets you compare the values of an OpenCL Kernel variable across work-items and work-groups.

Variable Values

The Multiwatch main window displays either a graphical (image view) or spreadsheet (data view) visualization of the selected kernel variable across the various work-items and work-groups. This view operates very similarly to the Object views. The two following screenshots display a multi watch view for the kernel variable sum.s2. The upper screenshot shows an image view, the lower shows a data view.

OpenCL Mult	i-Watch 3				x
Image view	Data view			Multiple Kernel W	ork Item Watch
		sum.s2		Variable Name:	sum.s2 V
				Variable Type:	N/A
	234		Viel	Global Work Offset:	N/A
11/2	Ner -	A Back Thomas	C B	Global Work Size:	N/A
			74	Local Work Size:	N/A
	(FI)	Star Marken	and a	Hovered W. Item:	X: 952, Y: 84
1	112		MET.	Selected W. Item:	N/A
		Performance and	11/1/	Hovered Value:	798
		and a star of the		Selected Value:	N/A
		Sandie A the		Hovered Color:	R: 199, G: 199, B: 199, A: 255
				Selected Color:	N/A
		JA 13		Place the mouse poi	nter over the image pixel
		ALG		to view the texel inf	
				Adjust variable value	e active range:
	Ye	E TABLES		0	1020
	10	THE MAN AN			
	N.	all and and		0 min	max 1020
	13				
	M				
	Ø		C		
5					
<			>		

nage vie	ew Data	view							> Multiple Kernel V	Vork Item Watch	
Link b	etween the I	image and Dat	a Views						Variable Name:	sum.s2	~
Show	values norm	alized to [02	55] range			Grid Zoon	ו: 🔤		Variable Type:	N/A	
Show	hexadecimal	values							Global Work Offset:	: N/A	
	X:3	<mark>X:4</mark>	X: 5	X:6	X:7	X: 8	X: 9	^	Global Work Size:	N/A	
f:0	415	420	449	470	466	461	440		Local Work Size:	N/A	
Y: 1	431	444	436	420	423	417	421		Hovered W. Item:		
Y: 2	470	479	446	405	390	362	362				
Y: 3	471	443	412	384	350	325	291		Selected W. Item:	N/A	
Y: 4	434	381	352	345	323	328	329		Hovered Value:	421	
Y: 5	420	393	368	362	350	361	356		Selected Value:	N/A	
Y: 6	411	417	403	401	389	388	375		Hovered Color:	R: 90, G: 90, B: 90, A: 255	
Y:7	374	380	383	398	388	381	344		Hovered color.	11 30/ 01 30/ 01 30/ 11 203	
Y: 8	366	377	383	373	340	302	266		Selected Color:	N/A	Ľ
Y: 9	393	375	372	393	381	368	335		Place the mouse po to view the texel in	inter over the image pixel	
Y: 10	464	457	455	424	401	369	359				
Y: 11	415	410	422	398	377	398	389		Adjust variable valu	le active range:	
Y: 12	460	446	414	419	425	460	463		309	762	
Y: 13	489	501	487	480	460	407	401				
Y: 14	462	489	499	467	441	463	475		0 min	ma	x 102
Y: 15	586	574	551	547	584	605	628				
Y: 16	578	576	608	638	661	643	606				
Y: 17	573	605	603	550	488	465	449				
Y: 18	495	450	443	461	492	548	573				
Y: 19	474	482	503	530	552	577	619				
v. 20	470	E/11	572	570	575	550	E26	~			

Variable Name Combo-box

Select a variable name from the list, or type a watch expression. If the expression can be parse and evaluated, the values are updated in the main view.

Kernel Work Geometry and Selected / Hovered Details

Selecting, or hovering with the cursor over, work-items displays

the kernel work geometry (local and global work size and offset) for the current N-dimensional kernel execution, as well as their location and value.

Value Range Slider

Shows the value range, and allows marking the valid value range. Values above and below these lines are colored to show they are not in the selected range.

 $[\]ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger > GPU Debugging Views > Visual Studio Native Debugging Views

Visual Studio Native Debugging Views

Many Visual Studio native views have information from the CodeXL GPU Debugging Engine.

Source Code Window

- During API debugging, CodeXL displays the C/C++ source that led to the API function call in the source code window.
- During OpenCL kernel debugging, CodeXL displays the kernel source code in the source code window. This is done from a project file, if available, or from a temporary file for applications that generate the kernel source at run time.
- Double-clicking an OpenCL kernel or program, or an OpenGL shader in the CodeXL Debugging Object Explorer, displays its source code in the source code window.
 - Double-clicking an object allocation call stack frame in the Memory view displays the source location associated with that stack frame in the source code window.

Breakpoints View

- CodeXL API function breakpoints are displayed as C/C++ function breakpoints.
- CodeXL kernel function name breakpoints are displayed as function breakpoints with the prefix Kernel: .
- CodeXL Error / Warning breakpoints appear as function breakpoints, with their descriptive string as the breakpoint.
- Kernel source code breakpoints are displayed as

breakpoints in the kernel source file. Note that if a temporary kernel source file is created at runtime, the breakpoints set in it do not work in future runs of the application. To associate a kernel with a source file, include the source file in the debugged project, with a .cl file extension.

Watch View

During OpenCL kernel debugging, enter variable names in the Watch view to see them update during debugging or when switching between work-items in the Work-Item toolbar.

Autos View

During OpenCL kernel debugging, the Autos view displays the values of variables near the program counter, if possible.

Locals View

During OpenCL kernel debugging, the Locals view displays all the available kernel variables in the current scope. An additional variable, KernelDispatchDetails, which indicates the kernel's work size and the selected work item ID and workgroup, is also displayed.

Call Stack View

During API debugging, the current API function's call stack is displayed in the Call Stack view. During OpenCL kernel debugging, the kernel's call stack is displayed in the Call Stack view. Note that since the kernel is debugged at execution time, the call stack does not contain the clEnqueueNDRangeKernel or clEnqueueTask API function call that started the debugging.





Help > Using CodeXL > GPU Debugger > GPU Debugging Views > Statistics View

Statistics View

This view lets you view statistical information about your OpenCL and OpenGL APIs usage.

Context Selection

Select an OpenCL or OpenGL context in the CodeXL Debugging Object Explorer to update the statistics to that context's information.

Graph View

This view contains a graphical representation of the information in the Function Types, Function Calls Statistics, State Change Statistics, Deprecated Function Statistics, and Vertex Batch Statistics views. Statistics - GL Context 1

Function Types Function Calls	s Depreca	ated Function	Calls Histor	y Vertex Batches
Function Type	% of Calls	# of Calls	% of Calls (# of Calls (Avg. per Frame)
Feedback Functions	0.00	0	0.00	0
Synchronization Functions	0.00	0	0.00	0
Buffer Functions	9.22	1,582	9.38	3
Query Functions	0.00	0	0.00	0
Name Functions	0.00	0	0.00	0
Matrix Functions	24.55	4,211	25.00	8
Texture Functions	21.51	3,690	21.88	7
Program and Shader Fun	0.00	0	0.00	0
Raster Functions	0.00	0	0.00	0
Draw Functions	21.46	3,682	21.88	7
Deprecated Functions	55.21	9,471	56.25	18
State Change Functions	42.97	7,371	43.75	14
🛕 Get Functions	7.90	1,356	6.25	2
Total		17,155		32
	Total			A
9000 -				1
700				Program and Shader Functions
7500			Fu	nctions that create, delete or manipulate the parameters of shad
5000		2253 - 33		d shading programs are considered Program and Shaders
1500				Inctions
3000				
	i i i i i i i i i i i i i i i i i i i			

x

Properties Box

This box, located in the lower right-hand side of the Statistics view, displays the selected object properties. The properties include information about the object, as well as an explanation of any warnings (represented by exclamation point icons next to the items).

Statistics Tabs

Statistics tabs include:

- **Function Type Statistics** Displays details of OpenCL / OpenGL function calls to categories. Note that a function can belong to multiple or none of the categories. To see which categories a function belongs to, find it in the Function Calls Statistics view.
- **Function Call Statistics** Displays a breakdown of all the OpenCL and OpenGL functions used, as well as useful tips and information about unrecommended functions.
- **Deprecated Function Statistics** Displays a breakdown of the usage of functions deprecated by any OpenCL or OpenGL version. In Analyze Mode, this view also displays details about usage of deprecated features in partially deprecated functions.
- **<u>Calls History</u>** Displays the history of API calls executed for last frame on the current displayed context.
 - <u>Vertex Batch Statistics</u> Displays a breakdown of vertex drawing OpenGL function calls (or vertex batches) by the number of vertices drawn in each.

 $[\]ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger > GPU Debugging Views > Statistics View > Function Type Statistics view

Function Type Statistics view

The Function Types Statistics view displays a breakdown of OpenCL / OpenGL function calls to categories.

Note that a function can belong to multiple or none of the categories. If you wish to know which categories does a function belong to, find it in the Function Calls Statistics view.

Statistics - GL Context 1

Function Types Function Call	s Depreca	ited Function	Calls Histo	ry Vertex Batches
Function Type	% of Calls	# of Calls	% of Calls	(, # of Calls (Avg. per Frame)
Feedback Functions	0.00	0	0.00	0
Synchronization Functions	0.00	0	0.00	0
Buffer Functions	9.22	1,582	9.38	3
Query Functions	0.00	0	0.00	0
Name Functions	0.00	0	0.00	0
Matrix Functions	24.55	4,211	25.00	8
Texture Functions	21.51	3,690	21.88	7
Program and Shader Fun	0.00	0	0.00	0
Raster Functions	0.00	0	0.00	0
Draw Functions	21.46	3,682	21.88	7
1 Deprecated Functions	55.21	9,471	56.25	18
State Change Functions	42.97	7,371	43.75	14
⚠ Get Functions	7.90	1,356	6.25	2
Total		17,155		32
	Total			
000 -				
				Program and Shader Functions
^{'500}				unctions that create, delete or manipulate the parameters of shade
000 -		2255 - 55		nd shading programs are considered Program and Shaders unctions
500				
000 -			in the second se	

x

Graph and Properties Views

Graph View

When in the Function Types Statistics view, the Graph view displays a bar graph showing the part each function type takes. The grid lines represent 25, 50, and 75 percent of all function calls. Selecting a function type in the list highlights it in the graph. Redundant State Change functions (when available) are marked in red; the Get functions are marked in orange.

Properties Box

When in the Function Types Statistics view, the Properties box shows information about the currently selected function type.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger > GPU Debugging Views > Statistics View > Function Call Statistics View

Function Call Statistics View

The Function Calls Statistics view shows the number of times each OpenCL / OpenGL function call was executed in the previously rendered frame, as well as its percentage of the total functions execution. Statistics - GL Context 1

Function Types Function Calls	s Depreca	ted Function	Calls Histor	y Vertex Batches	
Function Type	% of Calls	# of Calls	% of Calls (, # of Calls (Avg. per Frame)	1
Feedback Functions	0.00	0	0.00	0	
Synchronization Functions	0.00	0	0.00	0	i
Buffer Functions	9.22	1,582	9.38	3	
Query Functions	0.00	0	0.00	0	
Name Functions	0.00	0	0.00	0	
Matrix Functions	24.55	4,211	25.00	8	
Texture Functions	21.51	3,690	21.88	7	
Program and Shader Fun	0.00	0	0.00	0	
Raster Functions	0.00	0	0.00	0	
Draw Functions	21.46	3,682	21.88	7	
Deprecated Functions	55.21	9,471	56.25	18	
State Change Functions	42.97	7,371	43.75	14	
🛕 Get Functions	7.90	1,356	6.25	2	
Total		17,155		32	•
	Total				
9000 H				A	
7700				Program and Shader Functions	
7500			Fu	inctions that create, delete or manipulate the parameters of shad	ers
5000		2255 - 155		nd shading programs are considered Program and Shaders	
4500				lictoris	
3000 -					

x

Detailed Function Calls

For most of the functions, the calls to that function are displayed using a single entry regardless of the arguments passed to the function.

For the following functions, CodeXL provides a separation based on one of the arguments used in the function call to provide more precise information, i.e. they are listed by their enumerators.

 \cdot glBegin

- · glBindBuffer
- · glBindBufferARB
- glBindTexture
- · glDisable
- glDisableClientState
- glDrawArrays
- · glDrawArraysIndirect
- · glDrawArraysInstanced
- · glDrawArraysInstancedARB
- · glDrawArraysInstancedBaseInstance
- · glDrawArraysInstancedEXT
- · glDrawElements
- · glDrawElementsIndirect
- · glDrawElementsInstanced
- · glDrawElementsInstancedARB
- · glDrawElementsInstancedBaseInstance
- $\cdot glDrawElementsInstancedBaseVertexBaseInstance$
- · glDrawElementsInstancedEXT
- · glDrawRangeElements
- \cdot glDrawTransformFeedback
- $\cdot \ glDrawTransformFeedbackInstanced$
- $\cdot \ glDrawTransformFeedbackStream$
- · glDrawTransformFeedbackStreamInstanced
- glEnable
- glEnableClientState
- \cdot glIsEnabled
- · glMultiDrawArrays
- · glMultiDrawArraysEXT
- · glMultiDrawArraysIndirect
- · glMultiDrawElements
- · glMultiDrawElementsEXT
- glMultiDrawElementsIndirect
- · glTexParameter*

Exporting Function Calls Statistics Data Into a file

The Function Calls Statistics data can be exported to a file (.csv) using the right-click context menu "Export Function Calls Statistics" command. Exporting the function calls statistics data can help you compare the function calls statistics of different frames. It also allows you to perform regression tests by comparing the function calls statistics data of two versions of your application.

Functions Not Recommended

The Usage of certain OpenCL and OpenGL functions is unrecommended, mostly for performance taxing reasons. These functions are noted as such in this view in varying degrees mildly unrecommended (yellow warning sign), intermediately unrecommended (orange warning sign) and highly unrecommended (red warning sign). Click on an unrecommended function to display information about it and a better alternative to using it in the Properties view.

Graph and Properties Views

Graph View

When in the Function Calls Statistics view, the Statistics view Graph view will display a pie chart of the OpenCL / OpenGL function calls. Each pie "slice" is one API function (or in some cases, a combination of an OpenGL function and enumerator). Selecting a function or functions in the list causes their respective slice in the graph to be highlighted. The graph can be rotated by clicking on it and dragging.

Properties Box

When in the Function Calls Statistics view, the Statistics view Properties Box will display the function name. If the function is an unrecommended one, the Properties Box will also display an explanation of why using this function is unrecommended, as well as a recommendation for an alternative to using this

function.

 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger > GPU Debugging Views > Statistics View > Deprecated Function Statistics view

Deprecated Function Statistics view

The Deprecated Function Statistics view displays information about your usage of functions deprecated by any OpenCL or OpenGL version.

Function Types	Function Calls	Deprecated Function	Calls History	Vertex Bat	ches	
Function Name	Deprec	ation Reason	# of Calls	% of Calls	Deprecated at Ver	Removed at Version
\land glTexCoord2f	Comple	etely Deprecated Functi	2,104	12.26	OpenGL 3.0	OpenGL 3.1
⚠ glVertex3f	Comple	etely Deprecated Functi	2,104	12.26	OpenGL 3.0	OpenGL 3.1
🖞 glMatrixMode	Compl	etely Deprecated Functi	1,579	9.20	OpenGL 3.0	OpenGL 3.1
🋕 glLoadldentit	y Compl	etely Deprecated Functi	1,053	6.14	OpenGL 3.0	OpenGL 3.1
🛆 glBegin 🛛 🛛 🕅	oadIdentity ppl	etely Deprecated Functi	526	3.07	OpenGL 3.0	OpenGL 3.1
🖞 glEnd	Compl	etely Deprecated Functi	526	3.07	OpenGL 3.0	OpenGL 3.1
🖞 glOrtho	Comple	etely Deprecated Functi	526	3.07	OpenGL 3.0	OpenGL 3.1
🖞 glPopMatrix	Comple	etely Deprecated Functi	526	3.07	OpenGL 3.0	OpenGL 3.1
🏦 glPushMatrix	Comple	etely Deprecated Functi	526	3.07	OpenGL 3.0	OpenGL 3.1
🖞 glMultMatrix	Comple	etely Deprecated Functi	1	0.01	OpenGL 3.0	OpenGL 3.1
Total (10 iten	ns)		9,471	55.21		
			<u>/</u>	glLoadIde	entity	

Each of the lines in the list displays the usage of one Deprecated function or a combination of a partially deprecated function and a deprecated feature. The number of calls to this function or deprecated uses of this function and feature combination is displayed (as well as a percentage of the total API function calls). Finally, the OpenCL / OpenGL version when this feature was deprecated and the version it was removed (if any) are displayed.

Graph and Properties Views

Graph View

When in the Deprecated Function Statistics view, the Statistics view Graph view displays a pie chart, representing a breakdown of the deprecated function calls per combination of deprecated function and deprecated feature.

Properties View

When in the Deprecated Function Statistics view, the Statistics view Properties view displays the currently selected deprecated function's name, as well as information about the deprecated feature it is a part of, and other functions and behaviors belonging to the same deprecated feature. A forward-compatible alternative to this feature is supplied in bold font.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger > GPU Debugging Views > Statistics View > API Function Calls History View

API Function Calls History View

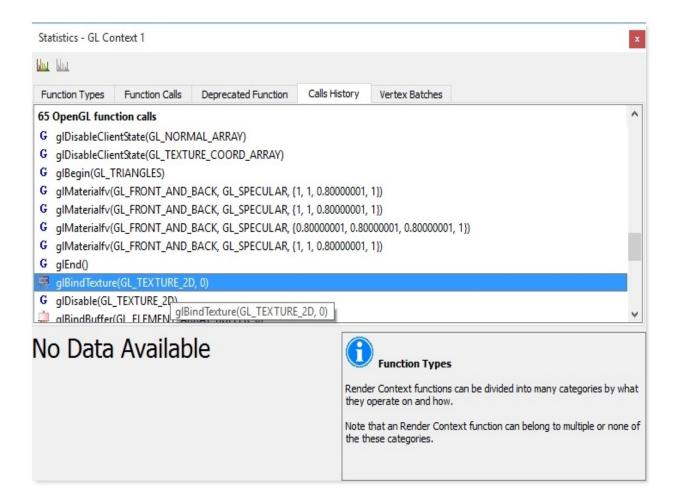
This view displays a log of OpenCL, OpenGL, OpenGL extensions, as well as WGL and glX function calls executed in each context.

Function Calls List

The function calls are displayed as a list, ordered by the time in which they were called. Each list line represents a single function call; it contains:

- the called function name,
- · its arguments values, and
- the function type (as an icon).

The function types are marked as shown in the following screenshot.



The icons have the following meanings

С	OpenCL function.
-	OpenCL buffer and image function.
1	OpenCL queue function.
G	OpenGL function.
Е	OpenGL extension function.
w	A WGL function.
x	A GLX function.
Đ	OpenGL program and shader function.
	OpenGL texture function.
1	1

—	OpenGL buffer function.
+	GL_string_marker_GREMEDY function.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger > GPU Debugging Views > Statistics View > Vertex Batch Statistics view

Vertex Batch Statistics view

The Vertex Batch Statistics view displays information about your usage of OpenGL vertex drawing functions or vertex batches, divided by the batch size (i.e. how many vertices were drawn with a single OpenGL function call).

<u>ha</u> <u>hha</u>						
Function Types	Function Calls	Deprecated Function	Calls History	Vertex Batch	nes	
Vertices per Batc	h	# of Batches	% of E	atches	# of Vertices	% of Vertices
1		16,368	26.67		16,368	0.03
<u>()</u> 6		123	0.20		738	0.00
1 24		8,061	13.13		193,464	0.38
10 72		4,092	6.67		294,624	0.58
1,536		32,736	53.33		50,282,496	99.01
lotal		61,380	100%		50,787,690	100%
			·····	72 Vertex	Batch	
28000						with 6 670% of your
32000 28000 24000			You	draw 0.58019		with 6.67% of your
28000			batc	draw 0.5801% hes	% of your vertices	
28000			batc	draw 0.58019 hes t is good practic	% of your vertices	with 6.67% of your vertices using as few batch the application's rendering

Vertex Batches

Each of the lines in the view displays statistics for a range of vertex batch sizes (normal view) or a specific batch size (detailed view). The number and percentage of batches of this size represents the "cost" of drawing with this batch size. The number and percentage of vertices represent the "benefit" gained from drawing with this batch size. Batches which are small in relation to the application (have a higher percentage of batches than is expected from their percentage of vertices) are noted with a warning icon.

Show Detailed Batch Statistics

Toggles between the Vertex Batch Statistics view's normal (gather batch sizes into ranges by amount) and detailed (show each vertex batch size separately) views.

Graph and Properties Views

Graph View

When in the Vertex Batch Statistics view, the Statistics view Graph view displays a bar chart, in which each bar is a range of batch sizes. The height of the bar is the number of batches drawn in this range, and the bar's color represents the range's position in the application's distribution (Red bars are the smallest batches and green bars are the largest ones). The grid lines represent 25, 50 and 75 percent of vertex drawing function calls. When the Vertex Batch Statistics shows the detailed view, this graph becomes a histogram of the vertex batch sizes.

Properties View

When in the Vertex Batch Statistics view, the Statistics view Properties view displays the currently selected batch size range, the percentage statistics for this range and a short explanation about vertex batches. $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger > GPU Debugging Views > Object Views

Object Views

The Object views - Image view and Data view - allow you to view all OpenCL buffers and images, and OpenGL textures, static buffers, VBOs (vertex buffer objects), FBOs (frame buffer objects), render buffers and pbuffers (pixel buffers).

Each object can be viewed both as an image in the Image view (except OpenCL Buffers and OpenGL VBOs) and as a spreadsheet containing the object raw data in the Data view. The object properties and parameters will appear in the properties view.

<u>Image View</u> <u>Data View</u>

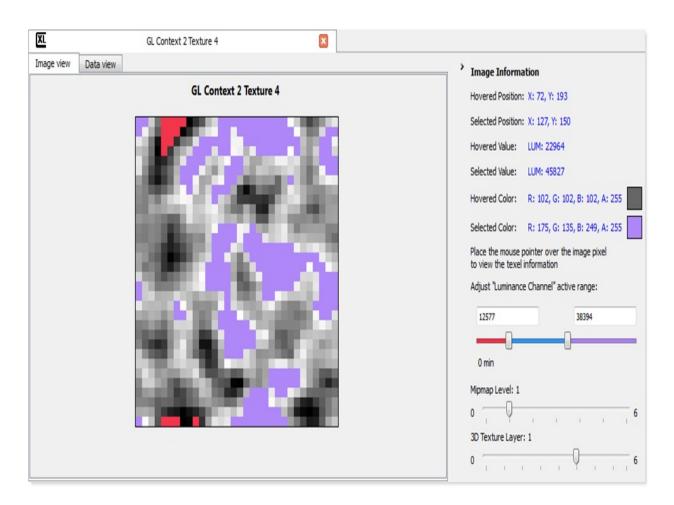
Information Panel

The information panel helps you make fast inquiries and adjustments to the currently viewed object pixels/texels.

The information panel is composed of a few main elements (which are only shown when relevant:

Current and Previous Pixel

When hovering with the mouse over the Image view, the currently highlighted pixel data is displayed as the current pixel; Both RGBA values and raw data value are displayed. If you wish to compare between two different pixels, you can save a specific pixel value by clicking on it with the left mouse button. The pixel data will be stored as the "Selected Pixel", allowing you to make fast comparison to another pixel.



Single Channel Range Adjustment Slider

When viewing an object that has a single component data format ("Depth" / "Luminance" / "Intensity" / etc.) the image view translates these values to grayscale values. Use the single channel range adjustment slider to adjust the data translation to grayscale values.

3D Layer Slider

When viewing 3D images or textures, you can scroll through the 2D images that make up the 3D image using the 3D layer slider located in the information panel.

Texture Array Layer Slider

When viewing texture arrays, you can scroll through the

textures that make up the array using the Texture array layer slider located in the information panel.

Texture Mipmap Slider

When viewing a texture which has automatic or manual mipmaps defined, you can scroll through the different texture levels using the Mipmap slider located inside the information panel.

OpenCL buffer / OpenGL VBO Format Controls

When viewing an OpenCL buffer or an OpenGL VBO, the information panel will contain several items that allow choosing how the VBO data is displayed:

- Data combo-box choose which kind of data the buffer contains. Each option in this combo-box matches either the OpenCL C types or an OpenGL gl*Pointer function ("Interleaved" matches glInterleavedArrays). Choose "All" to display all formats. An exception to this is the "Index" option which also matches the glDrawElements (and variants) function's indices parameter and not only glIndexPointer.
- Format combo-box choose the format the data is stored in, matching the OpenCL C type the kernel uses (__global XXXX*) size and type parameters of the gl*Pointer function or the format parameter of glInterleavedArrays. The formats are named as data-components-format, for example C4UB stands for color data stored in 4 components of unsigned byte type.
- Offset spin control choose the offset from the beginning of the VBO to the start of the data. This should match the pointer parameter of the gl*Pointer function.
- Stride spin control choose the stride (space in bytes between the end of one vertex's data and the next one's) to match the stride parameter of the gl*Pointer function.

XL	clutil.hpp	×	CL Context 4 CL Buffer	1 🔟	CL Context 4 CL Buffer 2			
ata view						> Buffer Display	Options	
				G	id Zoom:	Type:	OpenCL C	•
Show he	exadecimal values					Format:	int4	
	i	i	i	i	^	Offset:	0	-
0	61	92	113	255		Stride:		
1	113	173	210	510		June.	0	
2	173	260	308	765				
3	242	350	415					
4	313	441	533	1275				
5	385	536	659	1530				
6	452	631	778	1785				
7	508	719	881	2040				
8	581	813	994	2295				
9	646	899	1099	2550				
10	720	994	1213	2805				
11	803	1098	1336	3060				
12	870	1186	1443	3315	-			
13	939	1276	1550	3570				
14	1012	1370	1661	3825				
15	1099	1479	1784	4080				
16	1180	1584	1901	4335				
17	1253	1679	2010	4590				
18	1339	1786	2134	4845				
19	1419	1887	2256	5100				
20	1500	1988	2381	5355				
21	1590	2100	2516	5610				
22	1683	2218	2654	5865				
23	1770	2332	2787	6120				
24	1874	2462	2944	6375	v .			

© 2015 Advanced Micro Devices. All rights reserved.

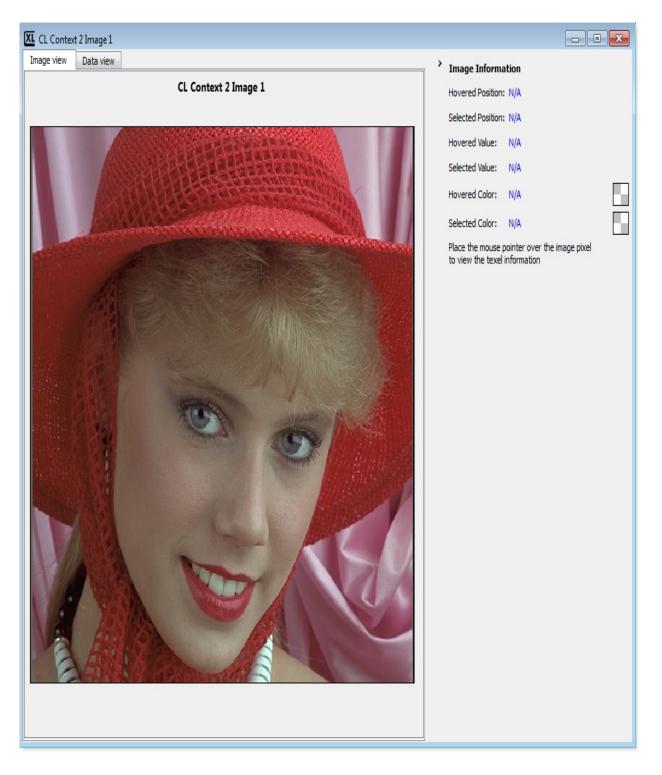




Help > Using CodeXL > GPU Debugger > GPU Debugging Views > Object Views > Object Image view

Object Image view

You can view an image, a buffer or a texture object loaded image using the Image view.



Images and Buffers toolbar

💫 🍳 🖑 🔞 🎯 🕲 🕲 🖉 🔛 S 🕹

Use the Images and Buffers toolbar to control the Image view

mage view	Data view					Image Information	
Link betwee	en the Image and [Data Views				Hovered Position: X: 5, Y: 576	
Show value	es normalized to [0.	.255] range		Gr	id Zoom: 🖳	Selected Position: N/A	
Show hexa	decimal values					Hovered Value:	
	0	1	2	3	4	Selected Value: N/A	
Red: 511	99	106	112	114	127	Hovered Color: R: 130, G: 65, B: 84, A: 255	
Green: 511	99	95	90	70	62		h
Blue: 511	99	99	97	85	82	Selected Color: N/A Place the mouse pointer over the image pixel	
Alpha: 511	255	255	255	255	255	to view the texel information	
Red: 510	99	106	112	116	130		
Green: 510	99	95	90	72	65	-	
Blue: 510	99	99	97	86	84	-	
Alpha: 510	255	255	255	255	255		
Red: 509	99	106	112	116	127		
Green: 509	99	95	90	72	65		
Blue: 509	99	99	97	86	83		
Alpha: 509	255	255	255	255	255		
Red: 508	99	106	113	115	127		
Green: 508	99	95	92	74	65		
Blue: 508	99	99	99	88	83		
Alpha: 508	255	255	255	255	255		
Red: 507	99	106	110	116	130		
Green: 507	99	95	91	75	68		
Blue: 507	99	99	97	89	86	• •	

Click on the data tab to see the image data

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger > GPU Debugging Views > Object Views > Object Data view

Object Data view

You can view a buffer, an image or a texture object raw data using the Data view.

				Grid Zoom:		Buffer Display	Ontions
Show	hexadecimal value			Grid Zoom:		Type:	OpenCL C
	i		i	i		Format:	Int4
)	61	92	113	255			LUC4
	113	173	210	510		Offset:	0
2	173	260	308	765		Stride:	0
3	242	350	415	1020			
, 1	252	349	420	1020			
5	272	363	449	1020			
5	272	371	470	1020			
о 7	2/5	369	4/6	1020			
B	268	372	461	1020			
9	261	363	440	1020			
10	268	363	435	1020			
	205	379	455	1020			
11	289	373	449	1020			
12	293	373	4451	1020			
13	293	376	448	1020			
14	292	381	448	1020			
15	310	398	458	1020			
16	314	403	450	1020			
17	327	416	400	1020			
18	320	408	473	1020			
20	320	404	480	1020			
20	337	421	506	1020			
22	344	432	520	1020			
23	351	445	531	1020			
25	374	474	563	1020			
	391	486	580	1020			
25	410	506	603	1020			
26	410			1020			
27	432	521	623	1020			
28							
29 30	442 434	540 527	626 611	1020	-		

Grid Zoom

The grid zoom slider allows you to set the zoom level of the displayed raw data cells.

Link Image and Data Views

When enabled, clicking on a pixel at the image view will select this pixel at the data view grid.

Show Normalized Values

The values showed in the grid are the original data values of the object as held by OpenCL and OpenGL. These values may be in various data formats such as OpenCL C types, or GL_FLOAT, GL_INT, GL_SHORT and GL_BYTE. When enabled, the data view will display the values in the grid normalized to the GL_BYTE type ([0..255] range), regardless of the original data type.

Viewing OpenCL buffers and OpenGL Vertex Buffer Objects (VBOs)

These buffers can only be shown in the Data View. After choosing the VBO's data, format, offset and stride; each vertex's data will be shown as a separate line in the view. The column headers represent the meaning of each value - X, Y, Z, W for vertex position; Nx, Ny, Nz for normal direction; R, G, B, A for vertex color and S, T, R, Q for texture coordinates.

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger > GPU Debugging Dialogs

GPU Debugging Dialogs

This section provides information on:

- Adding / removing breakpoints
- **<u>GPU debugging project settings</u>**
- **<u>GPU debugging global settings</u>**

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger > GPU Debugging Dialogs > Add / Remove CodeXL Breakpoints Dialog

Add / Remove CodeXL Breakpoints Dialog

The Breakpoint dialog lets you choose OpenCL and OpenGL API function breakpoints, as well as kernel function name breakpoints.

Breakpoints added this way are added as C/C++ function breakpoints to the Visual Studio Breakpoints view. Adding an OpenCL or OpenGL function as a Visual Studio breakpoint (Debug > Breakpoints > New Breakpoint...) adds it to this dialog.

The debugged process execution is suspended when a debugged process thread calls an API function marked as a breakpoint. The process run is suspended before the function is executed; this lets you observe the effect the breakpoint function has on the application behavior.

The debugged process execution is suspended when a kernel with a function name marked as a breakpoint starts executing. This lets you start kernel debugging without having to use the clEnqueueNDRangeKernel function call.

API Functions Kernel Functions Error / Warning	Active Breakpoints	
clEnqueueBarrierWithWaitList	Breakpoint Name	Туре
clEnqueueMarkerWithWaitList	ClEnqueueNDRangeKernel	cl API
clEnqueueWaitForEvents	✓ clFlush	cl API
clWaitForEvents	wglSwapBuffers	gl API
glClientWaitSync	glDrawElements	gl API
	ClCreateProgramWithSource	cl API
	Add >> Two keepel function name	cl API
	Add >> Type kernel function name	
	<< <u>R</u> emove	
	Remove All	
wait d	Enable all Breakpoints	

Breakpoints

API Functions	Contains OpenCL and OpenGL API funct supported by CodeXL.
Kernel Functions	When an OpenCL application is debugge
	 These are special breakpoints that can k Break on OpenGL error - This is hit wh error code. Note that if your applicatio Break on OpenCL error - This is hit wh

Error / Warning	 error code. Break on Detected error - The CodeXL specifications. This breakpoint is hit wl Break on Deprecated function - OpenG marked as deprecated and must be ren is called.
Text Filter	Enables filtering the Breakpoints list. Er containing all the strings.
Active Breakpoints	Contains the currently selected API and checking / unchecking the box next to th were not created yet in the application.
Add Breakpoint	To add a breakpoint, select one or more Breakpoints list by double-clicking on it,
Remove Breakpoint	To remove a breakpoint, select the funct pressing the Remove button.
Remove all Breakpoints	Press this button to remove all of the ac

Settings

Enable /Disable all Function Breakpoints

This section describes how to change the enabled status of all active breakpoints.

 $[\]ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger > GPU Debugging Dialogs > GPU Debugging Project settings

GPU Debugging Project settings

Use the File > Project Settings menu (Ctrl-P) to open the project settings dialog. Navigate to the "Debug" node in the project settings tree displayed in the left panel of the dialog.

CodeXL Project Settings		Х
General Debug ✓ Profile → CPU Profile Application Timeline Trace GPU Profile: Perf. Counters Power Profile Frame Analysis ✓ Analyze → Kernel/Shader Build Options	OpenGL Frame Terminators glClear_ glFlush glFinish SwapBuffers wglMakeCurrent SwapLayerBuffers glFrameTerminatorGREMEDY	
HLSL Build Options	OpenCL Frame Terminators	
	d gremedy_computation_frame dFlush dFinish dWaitForEvents	Add / Remove Breakpoints
Restore Default Settings		OK Cancel

OpenCL frame terminators are the functions that end your application computation

OpenCL Frame Terminators	frame. They let you view the log of a single computation frame, not the entire calls log. Available Frame Terminators are: cl_gremedy_computation_frame, clFlush, clFinish, and clWaitForEvents.
OpenGL Frame Terminators	OpenGL frame terminators are the functions that end your application render frame. They let you view the log of a single render frame, not the entire calls log. Available Frame Terminators are: glClear, glFlush, glFinish, wglSwapBuffers, wglMakeCurrent, wglSwapLayerBuffers, and glFrameTerminatorGREMEDY. For example: glFlush is usually chosen for single-buffered applications. wglSwapBuffers is usually chosen for double-buffered applications. You must select at least one OpenGL Frame Terminator. See Frames and Frame Terminators[A1] for more details.
Add / Remove Breakpoints	Click this button to open the <u>Add / Remove</u> <u>CodeXL Breakpoints dialog</u> .





Help > Using CodeXL > GPU Debugger > GPU Debugging Dialogs > GPU Debugging Global Settings

GPU Debugging Global Settings

The GPU Debug global settings are set when the project is initially created, and affect every session.

To set the options:

- 1. In the CodeXL menu, click **CodeXL Options**. The Edit CodeXL Global Settings dialog box is displayed.
- 2. Select the GPU Debug tab. See the descriptions below for each of the global debug settings.
- 3. Click **OK** to activate the new options and close the dialog box, or click **Restore Default Settings** to reset the dialog box selections to the system default, or click **Cancel** to close without changes.

General	GPU Debug	CPU Profile	Analyze	
Call Stat	ck			
	ect allocated obj	ects' creation ca	stacks	
HTML La	g File			
Enal	ble <mark>texture and</mark> i	mage logging in	he Calls History HTML log (recomm	iended)
Texture	s and images da	ta file format		PNG 🔻
* 3D tex	xtures and Image	es are always st	ored as tiff images	
* tiff an	d png formats p	reserve alpha ch	annel information	
Calls Log	gging			
Maximu	m amount of iten	ns logged:		
OpenGL	function calls (p	er context)		4000000 ≑
OpenCL	function calls (p	er context)		50000 🖨
* If this Define	maximum is exce a frame termina	eeded, the log w ator to avoid this	ill be cleared. automatic clearing.	
* the log	g will not be clea	red during a glBe	gin-glEnd block.	
Advance	ed			
Items sh	nown in monitore	d objects tree (j	er type)	200 🗘
Flus	h log file after ev	very monitored f	unction call (not recommended)	

Call Stack

Additional Source Code Directories	When specifying additional source code directories, If the source code file is not found at its debug information path, it will searched, by file name only, in the additional source code directories. The search is performed according to the order in which
	the directories are specified. The additional source code directories are semicolon separated.
	When specifying a source code root location,

Source Code Root Location	the root location is prefixed to each source code file path. Only one source code root location can be specified. Example: If the source code root is "D:\Dir1\" and the debug information source code path is "\Dir2\Dir3\MyFile.cpp", the Source Code editor input path will be "D:\Dir1\Dir2\Dir3\MyFile.cpp".
Collect Allocated Objects' Creation Calls Stacks	Deselect this checkbox if you don't want CodeXL to collect the graphic memory allocated objects creation calls stacks (which are displayed in the Memory View). This can improve Debug Mode performance in some cases when many graphic memory objects are allocated.

HTML Log File

Enable Texture Images Logging in the Calls History HTML Log	When this box is checked, textures data is saved and displayed in the calls log file.
Textures Data File Format	You can select the format by choosing the appropriate radio button. Available formats are: JPG, BMP, PNG, and TIFF. 3D textures are stored as tiff images.

Calls Logging

Set the logging limit for OpenGL API function calls and OpenCL

API function calls. If this limit is exceeded, the log is cleared. Define frame terminators in the project settings to avoid this automatic clearing.

Advanced

Floating-Point Precision	The maximum number of significant digits that will be displayed in the Object and Multi-Watch views.
Flush Log File After Every Monitored Function Call	When the Flush log file after every monitored function call check box is checked, CodeXL will flush the OpenCL / OpenGL calls history log file after every API function call instead of not use memory cached batches. This feature can help tracking the function call that led to a debugged application crash. Using this feature dramatically slows down the debugged application performance and therefore it is not recommended for regular use.
Restore Default Settings	Click this button to restore all settings on all pages to their default values.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger > GPU Debugging API Support

GPU Debugging API Support

CodeXL GPU debugging supports OpenCL up to and including version 2.0, OpenGL up to and including version 4.5 (including compatibility profiles) and a large variety of OpenCL and OpenGL extensions.

Kernel debugging is not supported for OpenCL 2.0 kernels. The supported extensions include:

OpenCL Extensions

Extension name
cl_khr_3d_image_writes
cl_khr_byte_addressable_store
cl_khr_dx9_media_sharing
cl_khr_d3d10_sharing
cl_khr_d3d11_sharing
cl_khr_fp64
cl_khr_gl_sharing
cl_khr_icd
cl_khr_global_int32_base_atomics
cl_khr_global_int32_extended_atomics
cl_khr_local_int32_base_atomics
cl_khr_local_int32_extended_atomics
cl_ext_atomic_counters_32
cl_ext_cl20_atomics
cl_ext_cl20_svm
cl_ext_device_fission
cl_amd_c11_atomics

cl_amd_device_attribute_query
cl_amd_fp64
cl_amd_media_ops
cl_amd_popcnt
cl_amd_printf
cl_amd_svm
cl_amd_vec3
cl_nv_d3d9_sharing
cl_nv_d3d10_sharing
cl_nv_d3d11_sharing
* Using this extension in a kernel might prevent the kernel from

OpenGL extensions

Extension name	Support level
GL_ARB_arrays_of_arrays	Standard
GL_ARB_base_instance	Full
GL_ARB_blend_func_extended	Standard
GL_ARB_clear_buffer_object	Standard
GL_ARB_color_buffer_float	Standard
GL_ARB_compatibility	Full
GL_ARB_compressed_texture_pixel_storage	Standard
GL_ARB_compute_shader	Full
GL_ARB_conservative_depth	Standard
GL_ARB_copy_buffer	Full
GL_ARB_copy_image	Standard
GL_ARB_debug_output	Full
GL_ARB_debug_output	Full
GL_ARB_depth_buffer_float	Standard
GL_ARB_depth_clamp	Full
GL_ARB_depth_clamp	Standard
GL_ARB_depth_texture	Standard
GL_ARB_draw_buffers	Standard
GL_ARB_draw_buffers_blend	Standard
GL_ARB_draw_elements_base_vertex	Full
1	

GL_ARB_draw_elemnts_base_vertex	Full
GL_ARB_draw_indirect	Full
GL_ARB_draw_instanced	Full
GL_ARB_ES2_compatibility	Standard
GL_ARB_ES3_compatibility	Standard
GL_ARB_explicit_attrib_location	Standard
GL_ARB_explicit_uniform_location	Standard
GL_ARB_fragment_coord_conventions	Full
GL_ARB_fragment_coord_conventions	Standard
GL_ARB_fragment_layer_viewport	Standard
GL_ARB_fragment_program	Standard
GL_ARB_fragment_program_shadow	Standard
GL_ARB_fragment_shader	Full
GL_ARB_framebuffer_no_attachments	Standard
GL_ARB_framebuffer_object	Full
GL_ARB_framebuffer_sRGB	Full
GL_ARB_geometry_shader4	Full
GL_ARB_geometry_shader4	Full
GL_ARB_get_program_binary	Standard
GL_ARB_gpu_shader_fp64	Full
GL_ARB_gpu_shader5	Full
GL_ARB_half_float_pixel	Standard
GL_ARB_half_float_vertex	Standard
GL_ARB_instanced_arrays	Full
GL_ARB_instanced_arrays	Full
GL_ARB_internalformat_query	Standard
GL_ARB_internalformat_query2	Standard
GL_ARB_invalidate_subdata	Standard
GL_ARB_map_buffer_alignment	Standard
GL_ARB_map_buffer_range	Standard
GL_ARB_matrix_palette	Standard
GL_ARB_multi_draw_indirect	Full
GL_ARB_multisample	Standard
GL_ARB_multitexture	Full
GL_ARB_occlusion_query	Standard
GL_ARB_occlusion_query2	Standard

GL_ARB_pixel_buffer_object	Standard
GL_ARB_point_parameters	Full
GL_ARB_point_sprite	Standard
GL_ARB_program_interface_query	Standard
GL_ARB_provoking_vertex	Full
GL_ARB_provoking_vertex	Standard
GL_ARB_robust_buffer_access_behavior	Standard
GL_ARB_sample_shading	Standard
GL_ARB_sampler_objects	Standard
GL_ARB_seamless_cube_map	Full
GL_ARB_seamless_cube_map	Standard
GL_ARB_separate_shader_objects	Standard
GL_ARB_shader_atomic_counters	Full
GL_ARB_shader_bit_encoding	Standard
GL_ARB_shader_image_load_store	Standard
GL_ARB_shader_image_size	Standard
GL_ARB_shader_objects	Full
GL_ARB_shader_precision	Standard
GL_ARB_shader_storage_buffer_object	Standard
GL_ARB_shader_subroutine	Standard
GL_ARB_shading_language_100	Full
GL_ARB_shading_language_420pack	Standard
GL_ARB_shading_language_packing	Standard
GL_ARB_shadow	Full
GL_ARB_shadow_ambient	Standard
GL_ARB_stencil_texturing	Standard
GL_ARB_sync	Full
GL_ARB_sync	Full
GL ARB tessellation shader	Full
GL ARB texture border clamp	Full
GL_ARB_texture_buffer_object	Full
GL_ARB_texture_buffer_object_rgb32	Standard
GL_ARB_texture_buffer_range	Standard
GL ARB texture compression	Full
GL ARB texture compression rgtc	Standard

GL_ARB_texture_cube_map	Full
GL_ARB_texture_cube_map_array	Full
GL_ARB_texture_env_add	Full
GL_ARB_texture_env_combine	Full
GL_ARB_texture_env_crossbar	Full
GL_ARB_texture_env_dot3	Full
GL_ARB_texture_float	Standard
GL_ARB_texture_gather	Standard
GL_ARB_texture_mirrored_repeat	Full
GL_ARB_texture_multisample	Full
GL_ARB_texture_multisample	Standard
GL_ARB_texture_non_power_of_two	Full
GL_ARB_texture_query_levels	Standard
GL_ARB_texture_rectangle	Full
GL_ARB_texture_rg	Standard
GL_ARB_texture_rgb10_a2ui	Standard
GL_ARB_texture_storage	Standard
GL_ARB_texture_storage_multisample	Standard
GL_ARB_texture_swizzle	Standard
GL_ARB_texture_view	Standard
GL_ARB_timer_query	Standard
GL_ARB_transform_feedback_instanced	Standard
GL_ARB_transform_feedback2	Standard
GL_ARB_transform_feedback3	Standard
GL_ARB_transpose_matrix	Full
GL_ARB_uniform_buffer_object	Full
GL_ARB_vertex_array_bgra	Full
GL ARB vertex array bgra	Standard
GL ARB vertex array object	Full
GL ARB vertex attrib 64bit	Standard
GL ARB vertex attrib binding	Standard
GL ARB vertex blend	Full
GL ARB vertex buffer object	Standard
GL ARB vertex program	Standard
GL ARB vertex shader	Full

GL_ARB_vertex_type_2_10_10_10_rev	Standard
GL_ARB_viewport_array	Standard
GL_ARB_window_pos	Full
GL_EXT_bgra	Full
GL_EXT_bindable_uniform	Full
GL_EXT_blend_logic_op	Full
GL_EXT_blend_minmax	Full
GL_EXT_blend_subtract	Full
GL_EXT_compiled_vertex_array	Full
GL_EXT_direct_state_access	Full
GL_EXT_draw_instanced	Full
GL_EXT_framebuffer_blit	Full
GL_EXT_framebuffer_multisample	Full
GL_EXT_framebuffer_object	Full
GL_EXT_geometry_shader4	Full
GL_EXT_multi_draw_arrays	Full
GL_EXT_packed_pixels	Full
GL_EXT_stencil_two_side	Full
GL_EXT_texture	Full
GL_EXT_texture_array	Full
GL_EXT_texture_buffer_object	Full
GL_EXT_texture_integer	Full
GL_EXT_texture_rectangle	Full
GL_EXT_texture_shared_exponent	Full
GL_EXT_texture3D	Full
GL_EXT_vertex_shader	Standard
GL_AMD_debug_output	Full
GL AMDX debug output	Full
GL_APPLE_aux_depth_stencil	Full
GL_APPLE_client_storage	Full
GL_APPLE_element_array	Full
GL_APPLE_fence	Standard
GL_APPLE_float_pixels	Full
GL_APPLE_flush_buffer_range	Standard
GL_APPLE_flush_render	Standard

GL_APPLE_object_purgeable	Standard
GL_APPLE_packed_pixels	Full
GL_APPLE_pixel_buffer	Standard
GL_APPLE_specular_vector	Full
GL_APPLE_texture_range	Standard
GL_APPLE_transform_hint	Full
GL_APPLE_vertex_array_object	Standard
GL_APPLE_vertex_array_range	Standard
GL_APPLE_vertex_program_evaluators	Full
GL_APPLE_ycbcr_422	Full
GL_ATI_draw_buffers	Standard
GL_ATI_fragment_shader	Standard
GL_ATI_text_fragment_shader	Standard
GL_GREMEDY_frame_terminator	Full
GL_GREMEDY_string_marker	Full
GL_HP_occlusion_test	Standard
GL_KHR_debug	Standard
GL_NV_fragment_program	Standard
GL_NV_fragment_program_option	Standard
GL_NV_fragment_program2	Standard
GL_NV_geometry_shader4	Full
GL_NV_occlusion_query	Standard
GL_NV_primitive_restart	Full
GL_NV_texgen_reflection	Full
GL_NV_texture_rectangle	Full
GL_NV_texture_shader	Standard
GL_NV_texture_shader3	Standard
GL_NV_vertex_program	Standard
GL_NV_vertex_program1_1	Standard
GL_NV_vertex_program2	Standard
GL NV vertex program2 option	Standard
GL_NV_vertex_program3	Standard
GL_OES_draw_texture	Standard
GL_SGIS_generate_mipmap	Full
GL SGIS texture border clamp	Full

GL SGIS texture edge clamp	Full
GL SGIS texture lod	Full
GL_SGIS_texture_select	Full
GL_SGIX_depth_texture	Full
GL_SGIX_interlace	Full
GL_SGIX_shadow	Full
GL_SGIX_shadow_ambient	Full
GLX_ARB_create_context	Full
GLX_ARB_create_context_profile	Full
GLX_ARB_fbconfig_float	Standard
GLX_ARB_framebuffer_sRGB	Full
GLX_ARB_get_proc_address	Full
GLX_ARB_multisample	Standard
GLX_SGI_video_sync	Standard
GLX_SGIX_fbconfig	Standard
WGL_AMD_gpu_association	Standard
WGL_ARB_buffer_region	Standard
WGL_ARB_create_context	Full
WGL_ARB_create_context_profile	Full
WGL_ARB_extensions_string	Full
WGL_ARB_framebuffer_sRGB	Full
WGL_ARB_make_current_read	Standard
WGL_ARB_multisample	Standard
WGL_ARB_pbuffer	Standard
WGL_ARB_pixel_format	Standard
WGL_ARB_pixel_format_float	Standard
WGL_ARB_render_texture	Standard
WGL_I3D_genlock	Standard
WGL_NV_gpu_affinity	Standard
WGL_NV_present_video	Standard
WGL_NV_swap_group	Standard
WGL_NV_video_out	Standard

Standard Extension Support Level

The Standard Extension Support Level enables one to log the calls and arguments of the extension functions, set breakpoints at the extension functions, and watch the values of the extension states variables.

Full Extension Support Level

In addition to supporting the Standard Extension Support Level features, the Full Extension Support Level enables one to view the extension-related data in the corresponding CodeXL views.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Debugger > GPU Kernel Debugging Support

GPU Kernel Debugging Support

CodeXL GPU debugging supports OpenCL kernel debugging. In order to step into a kernel, and start debugging, use one of the following methods:

- 1. Breakpoint in clEnqueueNDRangeKernel
- 2. Break in a kernel

Breakpoint in clEnqueueNDRangeKernel

- 1. Put a breakpoint in clEnqueueNDRangeKernel using the <u>breakpoints</u> dialog.
- 2. Once the breakpoint hits, click F11, or step-in button
- 3. The kernel source file is opened, and you can use stepover, step-in and step-out buttons.

Break in a kernel

- 1. Run your application.
- 2. After the kernel is created, break into the debugger.
- 3. Open the <u>breakpoints</u> dialog.
- 4. Navigate to the "Kernel Functions" tab
- 5. Select the kernel in which you want to break, and click the "Add button".
- 6. Click Ok.

- 7. Resume the debugged application.
- 8. Next time this kernel is executed, the kernel source file will open, and you will be able to use step-over, step-in and step-out buttons.

Kernel Debugging

Once CodeXL goes into kernel debugging mode, the kernel source code is opened. If the kernel was created with a string, a temporary file with the kernel source is created.

🚾 CodeXL Teapot (Kernel Debugging) - CodeXL De	ebug Mode					- 0	Х
File Edit View Debug Profile Frame Analy (1) (1) (1) (1) (1) (1) (1) (1) (1) (1) (1) (1) (1) (1)	sis <u>A</u> nalyze <u>T</u> ools <u>W</u> indow <u>H</u> } 🗣 🕻 🔽 👰 🗟 🥹 🗿 (Wavefront 1	▼ Work Item X: 0	∨ Y: 0	v	Z: 0 ~
CodeXL Explorer & X	CL Contex	kt 1 Kernel 1 - applySources	×				
	30 int index = getInd 31 float2 pos = (floa 32	<pre>t2)((float)coord.x, (float)coord. 0 if this grid point is inside th os - p->spoutCenter; outInvExtent.x; outInvExtent.y; (dot(normPos, normPos), 1.0f); ntensity of smoke as a probabilit defined by p->sourceCenter. Anyth >sourceCenter; outInvExtent.x;</pre>	/); = teaspout, 0.0	0 otherwise. centered around			~
 OpenCL Program 1 OpenCL Program 2 Kernel 1 - applySources OpenCL Program 3 OpenCL Program 4 OpenCL Program 5 OpenCL Program 6 OpenCL Program 7 	46 47 // Increase densit 48 s[index] += (p->Ks 49 t[index] = clamp(t 50 51 // Set velocity ve 52 u[index] = (p->sou 53 }	outinvExtent.y; limit * p->sourceDistributionBeta y and temperature by calculated i Dens * p->deltaTimeInSeconds * in [index] + p->KsTemp * p->deltaTim ctor everywhere inside the teapot rceVelocity * limit);	ntensity at th: tensity); eInSeconds * in	is point.		;));	
OpenCL Program 1 OpenCL Program 2 OpenCL Program 2 OpenCL Program 3 OpenCL Program 4 OpenCL Program 5 OpenCL Program 6 OpenCL Program 7 OpenCL Program 8 V	45 float intensity = 46 47 // Increase densit 48 s[index] += (p->Ks 49 t[index] = clamp(t 50 51 // Set velocity ve 52 u[index] = (p->sou 53 } <	<pre>limit * p->sourceDistributionBeta y and temperature by calculated i Dens * p->deltaTimeInSeconds * in [index] + p->KsTemp * p->deltaTim ctor everywhere inside the teapot rceVelocity * limit);</pre>	ntensity at th: tensity); eInSeconds * in spout.	is point. ntensity, p->KminTemp, p->K	maxTemp);	;));	>
 OpenCL Program 1 OpenCL Program 2 Kernel 1 - applySources OpenCL Program 3 OpenCL Program 4 OpenCL Program 5 OpenCL Program 6 OpenCL Program 7 	45 float intensity = 46 47 // Increase densit 48 s[index] += (p->Ks 49 t[index] = clamp(t 50 51 // Set velocity ve 52 u[index] = (p->sou 53 } <	<pre>limit * p->sourceDistributionBeta y and temperature by calculated i Dens * p->deltaTimeInSeconds * in [index] + p->KsTemp * p->deltaTim ctor everywhere inside the teapot rceVelocity * limit); Watch</pre>	ntensity at th: tensity); eInSeconds * in spout. B × Breakpoints	is point. ntensity, p->KminTemp, p->K	maxTemp);		₽x
OpenCL Program 1 OpenCL Program 2 OpenCL Program 2 OpenCL Program 3 OpenCL Program 4 OpenCL Program 5 OpenCL Program 6 OpenCL Program 7 OpenCL Program 8 V	45 float intensity = 46 47 // Increase densit 48 s[index] += (p->Ks 49 t[index] = clamp(t 50 51 // Set velocity ve 52 u[index] = (p->sou 53 } <	<pre>limit * p->sourceDistributionBeta y and temperature by calculated i Dens * p->deltaTimeInSeconds * in [index] + p->KsTemp * p->deltaTim ctor everywhere inside the teapot rceVelocity * limit);</pre>	ntensity at th: tensity); eInSeconds * in spout. Breakpoints e Name O Ope O clEr V manue	is point. ntensity, p->KminTemp, p->K	maxTemp);	Value {X:0, Y:0, Z:	Type struct int3 int float2

While debugging the kernel you can:

- 1. Put a breakpoint in any of the kernel source code lines.
- 2. Step-in, Step-out and Step-over the kernel source code lines.
- 3. See the values of variables in various views:

· Locals view

The locals view display a tree of the current local variables.

Locals				x
Name		Value {X:0, Y:0, Z:0	Туре	^
✓ Kernel	dispatch details	{}	struct	
Y ND	Range	{}	struct	
	Work dimensions	3	uint	
>	Global work size	{64, 64, 1}	uint3	
>	Local work size	{64, 4, 1}	uint3	
>	Global work offset	{0, 0, 0}	uint3	
>	Global work-groups	{1, 16, 1}	uint3	
	Number of work-groups	16	uint	
	Number of work-items	4096	uint	

Watch view

•

Type an expression, or drag an expression from the source code view to see the current variable value.

Name	Value {X:0, Y:0, Z:0}	Туре
imit	0.000000	float
coord	{0, 0, 1}	int3

Use the work item toolbar in order to change the currently viewed work item both in Watch view and Locals view.

Wavefront 1 🔹	Work Item X:	0 🔻	Y:	0	•	Z:	0
---------------	--------------	-----	----	---	---	----	---

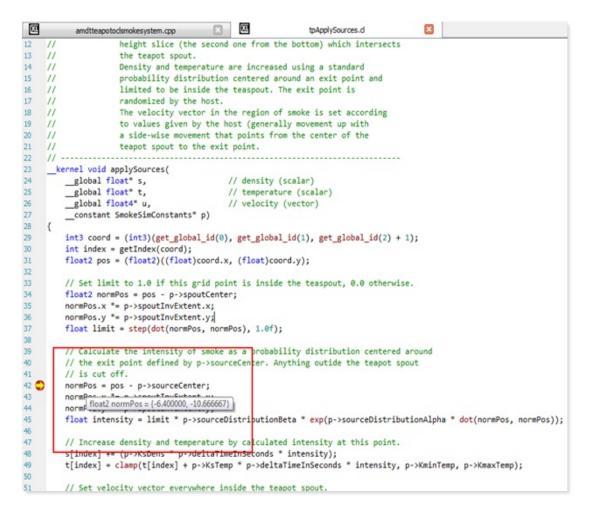
Multi-watch view

•

In order to see multiple work item view of a variable, open the <u>Multi-watch view</u> and use the visual and numeric data representation to see the variable's values.

Use the mouse to hover a variable and watch its value in a tooltip

As seen in the attached screenshot, once CodeXL is in Kernel Debugging mode, when the mouse cursor hovers over the variable name, a tooltip with the variable value is displayed.



 $\ensuremath{\textcircled{\sc c}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler

CPU Profiler

CodeXL's CPU Profiler is used for performance analysis and tuning of applications running on CPU.

The CPU Profiler lets you identify main performance bottlenecks of the profiled application or the entire system.

Features

- Three profile modes:
- Time-Based Profile (TBP) can identify the "hot-spots" in the profiled applications. (Hot-spots are code areas that use significantly more time compared to other areas in the code.)
- Event-Based Profile (EBP) can identify CPU and memory related performance issues in the profiled applications.
- Instruction-Based Sampling (IBS) can record and count the instructions that trigger HW events, as well as calculate various metrics, such as data cache latency.
- Per-Process mode and System-Wide mode profiling:
 - Per-process mode profiles a process and its children.
 - System-wide mode profiles the entire system.
- Attach to process for profiling.
- User-mode profiling and Kernel-mode profiling (Windows only).
- Native applications profiling (C, C++ and Fortran).
- · Profiling C++ inline functions.
- · CLR/.NET applications profiling (only on Windows).
- · Java applications profiling.
- · Call Stack Sampling (CSS) for all profile modes.
- Aggregation of the collected samples at various levels:

Process/Modules/Functions/Source and Disassembly.

- HW events counter multiplexing.
- Debugging Data Formats Supported:
 - CodeXL supports symbol information for unmanaged Executables compiled by MS Visual Studio or GCC (under Linux or other Unix-like systems (like Cygwin and MinGW)). That includes the following debugging data formats: PDB, COFF, DWARF, STABS.
 - For managed Executables, CodeXL supports Java and .NET applications' debug information.
- Time-Based Profile (TBP) and Event-Based Profile (EBP) are supported in guest OS running on VMware Workstation 11.0 or later.
- Time-Based Profile (TBP) and Event-Based Profile (EBP) are supported on Microsoft Hyper-V.
- Time-Based Profile (TBP) is supported on Xen Project hypervisor.
- Time-Based Profile (TBP) is supported on Linux KVM hypervisor.
- Controlling CPU Profiling i.e. pause and resume profiling, from target application to limit profiling scope.
- Aggregated instruction based (IMIX) report generation from CPU Profiler CLI.

Limitations

 CPU Profiler expects the profiled application executable binaries must not be compressed or obfuscated by any software protector tools, e.g. VMProtect.

Using the CPU Profiler

This section explains various key concepts related to CPU Profiling. It consists of the following subsections.

- **<u>CPU Profile Key Concepts</u>**
- **<u>CPU Profile Configurations</u>**

- **<u>CPU Profile Session</u>**
- CPU Profile Data Analysis
- **<u>CPU Profile Command Line Interface</u>**
- **<u>CPU Profile C/C++ Inline Functions</u>**
- **<u>CPU Profile PLT Relocations</u>**
- **<u>CPU Profile on Virtual Machine</u>**
- **<u>CPU Profile Control APIs</u>**
- **<u>CPU Profile IMIX report generation</u>**

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Key Concepts

CPU Profile Key Concepts

This section explains various key concepts related to CPU Profiling.

CPU Profiling in CodeXL

The CodeXL CPU Profiler follows a statistical sampling-based approach to gather the profile data periodically. It uses a variety of SW and HW resources available in AMD x86 based processor families. CPU Profiler uses the SW timer, HW Performance Monitor Counters (PMC), and HW IBS feature. The most time-consuming parts of a program have a larger number of samples; this is because they have a higher probability of being executed while samples are being taken by the CPU Profiler.

Sampling Interval

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

HW Performance Monitor Counters (PMC)

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

• In counting mode, these counters are used to count the specific events that occur in a CPU core.

In sampling mode, these counters are programmed to count a specific number of events; once the count is reached the appropriate number of times (called sampling interval), an interrupt is triggered. During the interrupt handling, the CPU Profiler collects profile data.

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

Time-Based Profile (TBP)

In this profile mode, the profile data is periodically collected based on the specified timer interval. It is used to identify the hot-spots of the profiled applications.

Event-Based Profile (EBP)

In this mode, the CPU Profiler uses the PMCs to monitor the various micro-architectural events supported by the AMD x86based processor. It helps to identify the CPU and memory related performance issues in profiled applications. CodeXL provides a number of predefined EBP profile configurations. To analyze a particular aspect of the profiled application (or system), a specific set of relevant events are grouped and monitored together. The CPU Profiler provides a list of predefined event configurations, such as Assess Performance and Investigate Branching, etc. You can select any of these predefine configurations to profile and analyze the runtime characteristics of your application. You also can create their custom configurations of events to profile.

This profile mode is supported on the various AMD processor

models, such as Family 10h, Family 11h, Family 12h, Family 14h, Family 15h models 00h-0Fh, 10-1Fh, 30-3Fh and Family 16h models 00-0Fh.

In this profile mode, a delay called skid occurs between the time at which the sampling interrupt occurs and the time at which the sampled instruction address is collected. This skid distributes the samples in the neighborhood near the actual instruction that triggered a sampling interrupt. This produces an inaccurate distribution of samples and events are often attributed to the wrong instructions.

Instruction-Based Sampling (IBS)

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

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

Event-Counter Multiplexing

If the number of monitored PMC events is less than, or equal to, the number of available performance counters, then each event can be assigned to a counter, and each event can be monitored 100% of the time. In a single-profile measurement, if the number of monitored events is larger than the number of available counters, the CPU Profiler time-shares the available HW PMC counters. (This is called **event counter multiplexing**.) It helps monitor more events and decreases the actual number of samples for each event, thus reducing data accuracy. The CPU Profiler auto-scales the sample counts to compensate for this event counter multiplexing. For example, if an event is monitored 50% of the time, the CPU Profiler scales the number of event samples by factor of 2.





Help > Using CodeXL > CPU Profiler > CPU Profile Configurations

CPU Profile Configurations

CodeXL handles all of the details and mechanics of data collection and profile formation.

CodeXL uses statistical sampling to collect and build a program profile. Because all profiles rely on statistical sampling, it is important for the CPU Profiler to take enough samples. The number of samples collected during a session depends on the following:

- \cdot $% \left(how frequently samples are taken (the sampling period or interval) \right)$
- the measurement length; that is, the amount of time which takes for the CPU Profiler to take a specific sample
- whether the profiling is system-wide

CodeXL provides the following profile sessions types:

Time-Based Profiling

Assess Performance

Instruction-Based Sampling

Investigate Branching

Investigate Data Access

Investigate Instruction Access

Investigate L2 Cache Access

Cache Line Utilization

Custom Profile

The type and frequency of the profiled events can indicate the presence of a pipeline bottleneck, poor memory access pattern,

poorly predicted conditional branches, or some other performance issues. Once hot-spots are found through timebased profiling, EBP and IBS are used identify opportunities for optimization. AMD processors provide a wide range of hardware events that can be monitored and measured. The number of counters and the hardware events that can be measured are processor-dependent.

When compiling your application to use for profiling, consider generating debug information as well. If present, the debug information can help identify profiled areas in the code; however, debug information is not required. Performance data can be collected for an application program that was compiled without debug information; in this case the results displayed by CodeXL are less descriptive. For example, CodeXL displays assembly code rather than function names or source code.

The CPU Profiler measures CPU execution time of a program. Through these measurements it helps you optimize the program. The Profiler records which functions were opened and how long it took to execute each one. CodeXL does this by taking samples. To take a sample, CodeXL interrupts the program's execution at specified intervals, and logs the state of the program's call chain.

CodeXL samples all of the various performance monitoring registers to obtain detailed information about the running application or the entire system. While all work can be done through the GUI, profiles also can be collected and analyzed through Visual Studio by using the CodeXL Visual Studio extension.

System profiling identifies a hot-spot anywhere in the system or an application under test. Any software component (an executable image, dynamically loaded library, device driver, or even the operating system kernel) that executes during the measurement period can be sampled. Any child processes spawned by a profiled process are profiled automatically. During the profiling process, the application to be analyzed is run at full speed on the same machine that is running CodeXL. Time-based samples (collected at 1 ms intervals on each core) can be used to identify possible bottlenecks, execution penalties, or optimization opportunities. The TBP feature can be used on both AMD and non-AMD processors with an Advanced Programmable Interrupt Controller (APIC) timer. Event-based samples and instruction-based samples can be used to help determine the cause of hot-spots or optimization opportunities. The sampling intervals are weighted so the types of view analysis are valid. The EBP and IBS features are only available on AMD processors.

Call chain sampling collects function call information, including caller-to-called relationships between functions. It is used in conjunction with the selected profile. When call chain sampling is enabled, CodeXL collects information from the run-time call stack of a monitored application process (and child processes) whenever a regular profile sample is taken for the process. When compared to other techniques, such as instrumentation, call chain sampling is a relatively low-overhead approach to the collection of function call information. However, call chain sampling results are subject to statistical variation.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Configurations > Time-Based Profiling

Time-Based Profiling

When time-based profiling is enabled and started, CodeXL configures a timer that periodically interrupts the program executing on a processor core.

When a timer interrupt occurs, a sample is created and saved for post-processing. Post-processing builds up a type of histogram, which describes what the system and its software components were doing. The most time-consuming parts of a program have the most samples because, most likely, the program is executing in those regions when a timer interrupt is generated and a sample is taken.

The frequency of sample taking is controlled by the timer interval. This sometimes is called the "sampling period." The timer interval is 1 millisecond: roughly 1,000 TBP samples are taken each second for each processor core.

The second factor is the length of time during which the samples are taken.. The measurement period depends on the overall execution time of the workload and the way in which CodeXL data collection is configured. Using the CPU Profile Options, CodeXL can be configured to collect samples for all, or part, of the time that the test workload is executing. If program run-time is short (less than 15 seconds), it may be necessary to increase program run-time by using a larger data set or more loop iterations to obtain a statistically useful result.

Deciding how many samples are enough requires a working knowledge about the characteristics of the workload under test. Scientific applications often have tight inner loops that are executed several times. In these situations, samples are being aggregated rapidly within the inner loops, and even a fairly short run-time yields a statistically useful number of samples. Other workloads, like transaction processing, usually have just a few inner loops, and the profiles are relatively "flat." For flat workloads, a longer measurement period is required to aggregate samples in code regions of interest.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Configurations > Assess Performance

Assess Performance

This pre-defined configuration is intended to get an overall assessment of hardware performance and to give an idea of the possible causes of performance issues.

Hardware Events: (the numeric hardware event codes are the codes the CPU uses to identify these events)

- [0C0] Retired Instructions
- [076] CPU clock cycles not halted
- [0C2] Retired branch instructions
- [0C3] Retired mispredicted branch instructions
- · [040] Data cache accesses
- · [041] Data cache misses
- · [046] L1 DTLB and L2 DTLB misses
- [047] Misaligned accesses

This profile configuration measures eight different events and requires event counter multiplexing. Each event is sampled approximately half of the time (a 50% duty cycle) when four hardware performance counters are available. When using a profile configuration that requires event counter multiplexing, ensure run time is long enough to build up a statistically accurate picture of program behavior.

The available views for profiles with this data are:

- All Data
- · Branch assessment
- · DTLB assessment
- · Data access assessment
- · IPC assessment
- Misaligned access assessment

· Overall assessment

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Configurations > Instruction-Based Sampling

Instruction-Based Sampling

Instruction-Based Sampling (IBS) identifies and diagnoses performance issues in program hot-spots. It collects data on how instructions behave on the processor and in the memory subsystem; it also provides a range of measurable data for each sample. When running IBS,

- hardware events are linked with the instructions that caused them.
- · it produces a wealth of event data in a single test run.
- latency is measured for key performance factors such as data cache miss latency.

IBS provides the most common types of information needed for program performance analysis. It uses a hardware sampling technique to generate event information similar to that produced by event-based profiling. Event-based profiling, however, offers a wider range of events that can be monitored, such as those related to HyperTransport[™] links.

Processor pipeline stages can be categorized into two main phases: instruction fetch and execution. Each instruction fetch operation produces a block of instruction data that is passed to the decode stages in the pipeline. The decoder identifies AMD64 instructions in the fetch block. These AMD64 instructions are translated to one or more macro-operations, called "macro-ops" or "ops," that are executed in the execution phase.

<u>Note:</u> For more information about instruction-based sampling, see the following documents which are available at <u>AMD's</u> <u>Developer Guides & Manuals page</u>:

- Software Optimization Guide for AMD Family 16h

Processors

- <u>Preliminary BIOS and Kernel Developer's Guide (BKDG)</u> for AMD Family 16h Models 00h-0Fh (Kabini) Processors
- <u>BIOS and Kernel Developer Guide (BKDG) for AMD</u> <u>Family 15h Models 00h-0Fh Processors</u>
- <u>Software Optimization Guide for AMD Family 15h</u> <u>Processors</u>
- <u>BIOS and Kernel Developer Guide (BKDG) for AMD</u> <u>Family 14h Models 00h-0Fh Processors</u>
- <u>BIOS and Kernel Developer's Guide (BKDG) For AMD</u> <u>Family 12h Processors</u>

How IBS Works

IBS provides separate means to sample fetch operations and macro-ops. IBS fetch sampling and IBS op sampling can be enabled and collected separately or together.

IBS Fetch Sampling

This is a statistical sampling method. IBS fetch sampling counts the completed fetch operations. When the number of completed fetch operations reaches the maximum fetch count (the sampling period), IBS tags the fetch operation and monitors that fetch operation until it either completes or aborts.

When a tagged fetch completes or aborts, a sampling interrupt is generated, and an IBS fetch sample is taken. An IBS fetch sample contains a timestamp, the identifier of the interrupted process, the virtual fetch address, and several event flags and values that describe what happened during the fetch operation. Similar to time-based profiling and event-based profiling, CodeXL uses the IBS sample data and information from the executable images, debug information, and source to build a profile IBS for software components executed on the system. IBS is also available in system-wide profiling.

The event data reported in an IBS sample includes the

following:

- If the fetch completed or aborted.
- If the address translation initially missed in the level one (L1) or level two (L2) instruction translation lookaside buffer (ITLB).
- The page size of the L1 ITLB address translation (4K, 2M).
- Whether the fetch initially missed in the instruction cache (IC).
- Fetch latency (number of processor cycles from when the fetch was initiated to when the fetch completed or aborted).

Event-based profiling requires several counters to collect as much information as IBS. The fetch address precisely identifies the fetch operation associated with the hardware events. The IBS fetch address may be the address of a fetch block, the target of a branch, or the address of an instruction that is the fall-through of a conditional branch. A fetch block does not always start with a complete, valid AMD64 instruction; this occurs when an AMD64 instruction straddles two fetch blocks. In this case, CodeXL associates the IBS fetch sample with the AMD64 instruction in the preceding fetch block.

A fetch can be abandoned before it delivers data to the decoder, or due to a control flow redirection; this can happen at any time during the fetch process. A fetch abandoned before initial access to the ITLB (before address translation) is not regarded as useful for analysis. These early abandoned fetches are called **killed** fetches.

CodeXL identifies killed fetches. The fetch operations remaining after killed fetches are removed from consideration are called **attempted** fetches: these fetches represent valid attempts to obtain instruction bytes.

A **completed** fetch is an attempted fetch that successfully delivered instruction data to the decoder. An **aborted** fetch is an attempted fetch that did not complete.

Note: Instruction fetch is an aggressive, speculative activity, and even instruction data produced by a completed fetch may not be used.

IbsOps IBS Op Sampling

IBS op sampling operates like fetch sampling. It provides two methods for op selection:

- Cycles mode IBS hardware counts processor cycles. When reaching the maximum cycle count (the sampling period), IBS tags an available valid op.
- Dispatched op mode IBS hardware counts ops as they are issued into the pipeline. When the number of dispatched ops reaches the maximum op count (the sampling period), IBS tags the op. Dispatched op mode is preferred because Cycles mode selection is susceptible to delay induced sampling bias.

<u>Note</u>: Some processors do not support dispatched op mode. For more details, see the <u>BKDG for the AMD processor</u> for your platform. The execution stages of the pipeline monitor the tagged macro-op. When the tagged macro-op retires, a sampling interrupt is generated, and an IBS op sample is taken. An IBS op sample contains:

- · a timestamp,
- \cdot $\;$ the identifier of the interrupted process,
- \cdot $% \$ the virtual address of the AMD64 instruction from which the op was issued, and
- several event flags and values that describe what happened when the macro-op executed.

CodeXL uses this and other information to build an IBS profile.

Cycle-based op sampling can be susceptible to timing bias: it can cause ops from some instructions to be selected more often than other instructions. Dispatched op-based sampling is the preferred IBS operating mode because it is not biased by timing. IBS op samples are taken only for ops that retire. Thus, IBS op event information does not measure speculative execution activity. The cycles-based tagging scheme can introduce statistical bias due to stalls at the decoding stage of the pipeline. If a macro-op is not available for tagging when the maximum op count is reached, the hardware tags a macro-op and starts counting again from a small, pseudo-random initial count.

IBS op sampling reports the following values for all ops:

- Virtual address of the parent AMD64 instruction from which the tagged op was issued.
- Tag-to-retire time (the number of processor cycles from when the op was tagged to when the op retired).
- Completion-to-retire time (the number of processor cycles from when the op completed to when the op was retired).

Attribution of event information is precise because the IBS hardware reports the address of the AMD64 instruction causing the events. For example, branch mispredictions are attributed to the mispredicted branch, and cache misses are attributed to the AMD64 instruction that caused the cache miss. IBS makes it easier to identify the performance-degrading instructions.

Some ops implement branch semantics. Branches include unconditional and conditional branches, subroutine calls, and subroutine returns.

Event information reported for branch ops include whether the branch was mispredicted or was taken.

IBS also indicates whether a branch operation was a subroutine return, and if the return was mispredicted. Some ops can perform a load (memory read), store (memory write), or a load and a store to the same memory address, as in the case of a read-op-write sequence.

When an op performs a load and/or store, event information includes the following:

• Whether a load was performed.

- Whether a store was performed.
- Whether address translation initially missed in the L1 and/or L2 data translation lookaside buffer (DTLB).
- Whether the load or store initially missed in the data cache (DC).
- · Virtual data address for the memory operation.
- Latency when a load misses the DC.

Requests made through the Northbridge produce additional event information:

- Whether the access was local or remote.
- Data source that fulfilled the request.

A full list of IBS op event information appears in the section on IBS-Derive events below. For hardware-level details, see the <u>BIOS and Kernel Developer's Guide</u> (BKDG) for the AMD processor for your platform.

IBS-Derived Events

CodeXL translates the IBS information produced by the hardware into **derived event** sample counts that resemble EBP sample counts. All IBS-derived events have "IBS" in the event name and abbreviation. Although IBS-derived events and sample counts look similar to EBP events and sample counts, the source and sampling basis for the IBS event information are different.

Arithmetic should **never** be performed between IBS derived event sample counts and EBP event sample counts. It is not meaningful to directly compare the number of samples taken for events that represent the same hardware condition. For example, fewer IBS DC miss samples is not necessarily better than a larger quantity of EBP DC miss samples.

Event	Description		
All IBS fetch samples	The number of all IBS fetch samples. This derived event counts the number of all IBS fetch samples that were		

	collected including IBS-killed fetch samples
IBS fetch killed	The number of IBS sampled fetches that were killed fetches. A fetch operation is killed if the fetch did not reach ITLB or IC access. The number of killed fetch samples is not generally useful for analysis and are filtered out in other derived IBS fetch events (except Event Select 0xF000 which counts all IBS fetch samples including IBS killed fetch samples.)
IBS fetch attempted	The number of IBS sampled fetches that were not killed fetch attempts. This derived event measures the number of useful fetch attempts and does not include the number of IBS killed fetch samples. This event should be used to compute ratios such as the ratio of IBS fetch IC misses to attempted fetches. The number of attempted fetches should equal the sum of the number of completed fetches and the number of aborted fetches.
IBS fetch completed	The number of IBS sampled fetches that completed. A fetch is completed if the attempted fetch delivers instruction data to the instruction decoder. Although the instruction data was delivered, it may still not be used (e.g., the instruction data may have been on the "wrong path" of an incorrectly predicted branch.)
	The number of IBS sampled fetches that aborted. An attempted fetch is aborted

IBS fetch aborted	if it did not complete and deliver instruction data to the decoder. An attempted fetch may abort at any point in the process of fetching instruction data. An abort may be due to a branch redirection as the result of a mispredicted branch. The number of IBS aborted fetch samples is a lower bound on the amount of unsuccessful, speculative fetch activity. It is a lower bound since the instruction data delivered by completed fetches may not be used.
IBS ITLB hit	The number of IBS attempted fetch samples where the fetch operation initially hit in the L1 ITLB (Instruction Translation Lookaside Buffer).
IBS L1 ITLB misses (and L2 ITLB hits)	The number of IBS attempted fetch samples where the fetch operation initially missed in the L1 ITLB and hit in the L2 ITLB.
IBS L1 L2 ITLB miss	The number of IBS attempted fetch samples where the fetch operation initially missed in both the L1 ITLB and the L2 ITLB.
IBS instruction cache misses	The number of IBS attempted fetch samples where the fetch operation initially missed in the IC (instruction cache).
IBS instruction cache hit	The number of IBS attempted fetch samples where the fetch operation initially hit in the IC.
	The number of IBS attempted fetch

IBS 4K page translation	samples where the fetch operation produced a valid physical address (i.e., address translation completed successfully) and used a 4-KByte page entry in the L1 ITLB.
IBS 2M page translation	The number of IBS attempted fetch samples where the fetch operation produced a valid physical address (i.e., address translation completed successfully) and used a 2-MByte page entry in the L1 ITLB.
IBS fetch latency	The total latency of all IBS attempted fetch samples. Divide the total IBS fetch latency by the number of IBS attempted fetch samples to obtain the average latency of the attempted fetches that were sampled.
IBS fetch L2 cache miss	The instruction fetch missed in the L2 Cache.
IBS ITLB refill latency	The number of cycles when the fetch engine is stalled for an ITLB reload for the sampled fetch. If there is no reload, the latency will be 0.
All IBS op samples	The number of all IBS op samples that were collected. These op samples may be branch ops, resync ops, ops that perform load/store operations, or undifferentiated ops (e.g., those ops that perform arithmetic operations, logical operations, etc.). IBS collects data for retired ops. No data is collected for ops that are aborted due to pipeline flushes, etc. Thus, all sampled ops are architecturally significant and

	contribute to the successful forward progress of executing programs.
IBS tag-to-retire cycles	The total number of tag-to-retire cycles across all IBS op samples. The tag-to- retire time of an op is the number of cycles from when the op was tagged (selected for sampling) to when the op retired.
IBS completion-to- retire cycles	The total number of completion-to-retire cycles across all IBS op samples. The completion-to-retire time of an op is the number of cycles from when the op completed to when the op retired.
IBS branch op	The number of IBS retired branch op samples. A branch operation is a change in program control flow and includes unconditional and conditional branches, subroutine calls and subroutine returns. Branch ops are used to implement AMD64 branch semantics.
IBS mispredicted branch op	The number of IBS samples for retired branch operations that were mispredicted. This event should be used to compute the ratio of mispredicted branch operations to all branch operations.
IBS taken branch op	The number of IBS samples for retired branch operations that were taken branches.
IBS mispredicted taken branch op	The number of IBS samples for retired branch operations that were mispredicted taken branches.

IBS return op	The number of IBS retired branch op samples where the operation was a subroutine return. These samples are a subset of all IBS retired branch op samples.
IBS mispredicted return op	The number of IBS retired branch op samples where the operation was a mispredicted subroutine return. This event should be used to compute the ratio of mispredicted returns to all subroutine returns.
IBS resync op	The number of IBS resync op samples. A resync op is only found in certain microcoded AMD64 instructions and causes a complete pipeline flush.
IBS all load store ops	The number of IBS op samples for ops that perform either a load and/or store operation. An AMD64 instruction may be translated into one ("single fastpath"), two ("double fastpath"), or several ("vector path") ops. Each op may perform a load operation, a store operation or both a load and store operation (each to the same address). Some op samples attributed to an AMD64 instruction may perform a load/store operation while other op samples attributed to the same instruction may not. Further, some branch instructions perform load/store operations. Thus, a mix of op sample types may be attributed to a single AMD64 instruction depending upon the ops that are issued from the AMD64 instruction and the op types.

IBS load ops	The number of IBS op samples for ops that perform a load operation.
IBS store ops	The number of IBS op samples for ops that perform a store operation.
IBS L1 DTLB hit	The number of IBS op samples where either a load or store operation initially hit in the L1 DTLB (data translation lookaside buffer).
IBS L1 DTLB misses L2 hits	The number of IBS op samples where either a load or store operation initially missed in the L1 DTLB and hit in the L2 DTLB.
IBS L1 and L2 DTLB misses	The number of IBS op samples where either a load or store operation initially missed in both the L1 DTLB and the L2 DTLB.
IBS data cache misses	The number of IBS op samples where either a load or store operation initially missed in the data cache (DC).
IBS data cache hits	The number of IBS op samples where either a load or store operation initially hit in the data cache (DC).
IBS misaligned data access	The number of IBS op samples where either a load or store operation caused a misaligned access (i.e., the load or store operation crossed a 128-bit boundary).
IBS bank conflict on load op	The number of IBS op samples where either a load or store operation caused a bank conflict with a load operation.
IBS bank conflict on	The number of IBS op samples where

store op	either a load or store operation caused a bank conflict with a store operation.
IBS store-to-load forwarded	The number of IBS op samples where data for a load operation was forwarded from a store operation.
IBS store-to-load cancelled	The number of IBS op samples where data forwarding to a load operation from a store was cancelled.
IBS UC memory access	The number of IBS op samples where a load or store operation accessed uncacheable (UC) memory.
IBS WC memory access	The number of IBS op samples where a load or store operation accessed write combining (WC) memory.
IBS locked operation	The number of IBS op samples where a load or store operation was a locked operation.
IBS MAB hit	The number of IBS op samples where a load or store operation hit an already allocated entry in the Miss Address Buffer (MAB).
IBS L1 DTLB 4K page	The number of IBS op samples where a load or store operation produced a valid linear (virtual) address and a 4-KByte page entry in the L1 DTLB was used for address translation.
IBS L1 DTLB 2M page	The number of IBS op samples where a load or store operation produced a valid linear (virtual) address and a 2-MByte page entry in the L1 DTLB was used for address translation.

IBS L1 DTLB 1G page	The number of IBS op samples where a load or store operation produced a valid linear (virtual) address and a 1-GByte page entry in the L1 DTLB was used for address translation.
IBS L2 DTLB 4K page	The number of IBS op samples where a load or store operation produced a valid linear (virtual) address, hit the L2 DTLB, and used a 4 KByte page entry for address translation.
IBS L2 DTLB 2M page	The number of IBS op samples where a load or store operation produced a valid linear (virtual) address, hit the L2 DTLB, and used a 2-MByte page entry for address translation.
IBS L2 DTLB 1G page	The number of IBS op samples where a load or store operation produced a valid linear (virtual) address, hit the L2 DTLB, and used a 1-GByte page entry for address translation.
IBS data cache miss load latency	The total DC miss load latency (in processor cycles) across all IBS op samples that performed a load operation and missed in the data cache. The miss latency is the number of clock cycles from when the data cache miss was detected to when data was delivered to the core. Divide the total DC miss load latency by the number of data cache misses to obtain the average DC miss load latency.
IBS load resync	Load Resync.
	The number of IBS op samples where a

IBS Northbridge local	load operation was serviced from the local processor. Northbridge IBS data is only valid for load operations that miss in both the L1 data cache and the L2 data cache. If a load operation crosses a cache line boundary, then the IBS data reflects the access to the lower cache line.
IBS Northbridge remote	The number of IBS op samples where a load operation was serviced from a remote processor.
IBS Northbridge local L3	The number of IBS op samples where a load operation was serviced by the local L3 cache.
IBS Northbridge local core L1 or L2 cache	The number of IBS op samples where a load operation was serviced by a cache (L1 data cache or L2 cache) belonging to a local core which is a sibling of the core making the memory request.
IBS Northbridge local core L1, L2, L3 cache	The number of IBS op samples where a load operation was serviced by a remote L1 data cache, L2 cache or L3 cache after traversing one or more coherent HyperTransport links.
IBS Northbridge local DRAM	The number of IBS op samples where a load operation was serviced by local system memory (local DRAM via the memory controller).
IBS Northbridge remote DRAM	The number of IBS op samples where a load operation was serviced by remote system memory (after traversing one or more coherent HyperTransport links and through a remote memory

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

 $\ensuremath{\textcircled{\sc c}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Configurations > Investigate Branching

Investigate Branching

This pre-defined configuration is intended to assist an investigation into branching and near-return performance.

Hardware Events: (the numeric hardware event codes are the codes the CPU uses to identify these events)

- · [0C0] Retired Instructions
- [0C2] Retired branch instructions
- · [0C3] Retired mispredicted branch instructions
- · [0C4] Retired taken branch instructions
- · [0C8] Retired near returns
- [0C9] Retired mispredicted near returns
- [0CA] Retired mispredicted indirect branches

The available views for profiles with this data are:

- · All Data
- · Branch assessment
- Near return report
- · Taken branch report

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Configurations > Investigate Data Access

Investigate Data Access

This pre-defined configuration helps investigate data locality and poor DTLB behavior.

Hardware Events: (the numeric hardware event codes are the codes the CPU uses to identify these events)

- [0C0] Retired Instructions
- · [040] Data cache accesses
- [041] Data cache misses
- · [042] Data cache refills from L2 or Northbridge
- · [045] L1 DTLB miss and L2 DTLB hit
- · [046] L1 DTLB and L2 DTLB misses
- · [047] Misaligned accesses

The available views for profiles with this data are:

- · All Data
- · DTLB assessment
- · DTLB report
- · Data access assessment
- · Data access report
- Misaligned access assessment

 $[\]ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Configurations > Investigate Instruction Access

Investigate Instruction Access

This pre-defined configuration helps investigate instruction fetches with poor L1 locality and poor ITLB behavior.

Hardware Events: (the numeric hardware event codes are the codes the CPU uses to identify these events)

- [0C0] Retired Instructions
- · [080] Instruction Cache fetches
- [081] Instruction Cache misses
- · [084] L1 ITLB miss and L2 ITLB hits
- · [085] L1 ITLB miss and L2 ITLB miss

The available views for profiles with this data are:

- All Data
- · ITLB report
- · Instruction Cache Report

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Configurations > Investigate L2 Cache Access

Investigate L2 Cache Access

This pre-defined configuration helps investigate memory access operations with poor L2 cache locality.

Hardware Events: (the numeric hardware event codes are the codes the CPU uses to identify these events)

- · [0C0] Retired Instructions
- · [07D] Requests to L2 cache
- [07E] L2 cache misses
- · [07F] L2 fill/writeback

The available views for profiles with this data are:

- All Data
- · L2 Access Report

 $[\]ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Configurations > Cache Line Utilization

Cache Line Utilization

This feature is a first step towards providing data-centric application profiling capabilities. This feature models the behavior of the processor L1 data cache and uses the Load and Store Instruction-Based Sampling (IBS) records to provide a measure of how efficiently an application utilizes the L1 data cache.

A cache is a relatively small amount of on-chip memory which is extremely fast compare to main memory. When the processor needs to access a location in main memory, it first checks whether a copy of that data is in the cache. If the data is present in cache, it is called a cache hit and the processor immediately accesses the data from cache. If the data is not in the cache, it is called a cache miss. In this case processor has to wait for the data to be fetched from main memory before it can continue to execute. All of the data required by all of the processes running on a processor cannot simultaneously fit in the cache, so the processor removes, or evicts, data from the cache when new data is needed and the cache is full. Data is transferred between memory and cache in blocks of fixed size, called cache lines.

The cache misses directly influence the performance of the application. Having the data in the cache when the processor needs it is one way to optimize performance of an application. Additionally, because cache size is small, it is desirable to fill the cache with data that will be used before it is evicted from the cache.

AMD processors have a separate instruction and data cache per core (L1 (Level 1) instruction and L1 data caches) as well as a unified L2 (per module) cache and L3 (per-chip) cache. However, CLU models the L1 data cache only. CLU measures how much of a cache line is used (read or written) before it is evicted from the cache. The percent cache line utilization is defined as percentage of number of bytes in the cache line had been accessed before the cache line has been evicted.

A low CLU value implies that the cache is being filled with data that is never or less accessed before it is evicted, implying cache capacity pressure, as well as main memory bandwidth pressure (reading data from main memory that is not accessed before being evicted).

A high usage percentage (CLU) means that the application is properly exploiting spatial and temporal locality of its data. Ideally, one would like to have 100 percent CLU. Practically speaking however, a good CLU is about 20 to 30 percent, primarily due to the sampling nature of the core in its collection of the load and store data.

Note: See the <u>BIOS and Kernel Developer's Guide for AMD</u> Family 10h Processors (order #31116) for detailed information about caches.

Event	Description
Cache Line Utilization Percentage	The cache line utilization percentage for all cache lines on all cores accessed by this instruction / function / module.
Line Boundary Crossings	The number of accesses to the cache line that spanned two cache lines. This happens when an unaligned access is made that causes two cache lines to be touched.
Bytes/L1 Eviction	The number of bytes accessed between cache line evictions.
	The number of accesses (loads plus

The following table describes the data that can be shown for each module, function, source and disassembly

Accesses/L1 Eviction	stores) to a cache line between evictions.
L1 Evictions	The number of times a cache line was evicted where this instruction depended on the data in the cache line.
Accesses	The total number of loads and stores samples for this instruction / function / module.
Bytes Accessed	The total number of bytes accessed by this instruction / function / module.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Configurations > Custom Profile

Custom Profile

This configuration is intended for advanced users who know what hardware events or combinations of events are important for specific analysis.

It allows a combination of hardware events, timer samples, and instruction-based sampling events. At least one event must be monitored for CodeXL to make the profiling.

Use of a smaller sampling period increases data collection overhead. Since data collection must be performed on the same platform as the test workload, more frequent sampling increases the intrusiveness of event-based profiling, and the sampling process adversely affects shared hardware resources such as instruction and data caches, translation lookaside buffers, and branch history tables. Extremely small sampling periods also can cause system instability. Start off conservatively and slowly decrease the sampling period for an event until the appropriate volume of samples is generated.

Warning You can demand too much sampling, causing the system to hang or crash. If this occurs, the easiest solution is to increase the sample intervals for the most popular events.

A factor when choosing the sampling period for an event is the workload behavior. Some workloads are CPU-intensive; other workloads are memory-intensive. Some workloads can be CPUintensive and require high memory bandwidth to stream data into the CPU. For example, a CPU-intensive application that does not accesses memory very often causes relatively few data-cache miss events. The characteristics of the workload can even vary by phase, where the phase setting up a computation has a different behavior from the computation phase itself. Thus, the workload behavior determines the frequency of certain kinds of events, requiring changes to the sample period.

The available views for the profile data depend on the events selected. The events are selected through the <u>Ref371333729</u> Dialog.

 $[\]ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Session

CPU Profile Session

A CodeXL CPU profile session collects profile data of a single execution of the profiled application. The profile session results can be viewed and analyzed using CodeXL tabbed views. After making the recommended changes to the profiled application, you can execute the same application and analyze the new bottlenecks after the change.

To create and run a CPU profile session in CodeXL requires **Executing CPU Profile Session** Other options can be configured in the project and global settings:

- 1. <u>Changing CPU Profiling Global Settings</u>
- 2. <u>Changing CPU Profiling Project Settings</u>
- 3. <u>Changing CPU Profiling Configuration Events</u>

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.



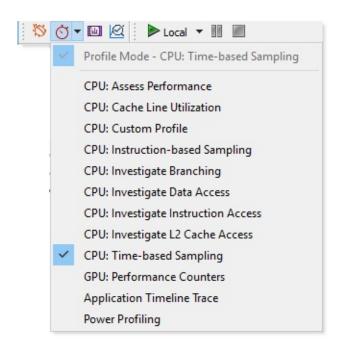


Help > Using CodeXL > CPU Profiler > CPU Profile Session > Execute a CPU Profile Session

Execute a CPU Profile Session

To run a CPU Profile session:

1. Open or create a CodeXL project



- Select a CPU profile type. See detailed information for each of CodeXL supported <u>CPU Profile Types</u> In the Active Mode toolbar, click the Profile Mode toolbar button to change the mode to Profiling (Profile > Profile Mode). Select any CodeXL CPU type of profile (Profile > CPU: Assess Performance for example).
- 3. Use the <u>**CPU Profile Setting dialog**</u> to configure the profile session parameters.
- Click the Start CodeXL Profiling toolbar button, ▶, to start profiling.
- 5. Optional: Pause / Stop the data collection of the profiled

application using the execution toolbar buttons: **II**. Stopping/pausing the profile session is optional. You can let the application run to its end and then the profiling session will automatically end. For long running applications, pausing allows you to control when the profiling data collection occurs so it matches the stage in your application execution that you want to profile.

6. When the profiled application execution is over, CodeXL processes the data collected, and a profile session window is opened. See the <u>View and Analyze the Profile</u> <u>Session Data Analysis</u> section.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Session > Attach to Process

Attach to Process

To start a CPU Profile session of an already running process:

- Select a CPU profile type. See detailed information for each of CodeXL supported <u>CPU Profile Types.</u> In the Active Mode toolbar, click the Profile Mode toolbar button to change the mode to Profiling (Profile > Profile Mode). Select any CodeXL CPU type of profile (Profile > CPU: Assess Performance, for example).
- 2. Use the <u>CPU Profile Setting dialog</u> to configure the profile session parameters. Note that if a CodeXL project is not opened when changing the profile settings, then a generic new project is automatically created.
- 3. Open the "Attach to Process..." dialog from the menu bar (Profile > Attach to Process...). Note that if a CodeXL project is not opened (or created) when the profiling session starts, then a generic new project is automatically created.

Prof	file	Frame Analysis	Analyze	Tools	Window
	Sta	rt Profiling (AMD)	[ClassicMat	Mul.exe)) F5
	Att	tach to Process			
~	Pro	ofile Mode - CPU:	Time-based	d Sampli	ng
88	Pau	use Data Collectio	n		F6
	СР	U: Assess Perform	ance		
	СР	U: Cache Line Util	ization		
	СР	U: Custom Profile			
	СР	U: Instruction-bas	ed Samplin	g	
	СР	U: Investigate Brai	nching		
	CPU: Investigate Data Access				
	CPU: Investigate Instruction Access				
_	CPU: Investigate L2 Cache Access				
~	CPU: Time-based Sampling				
	GPU: Performance Counters				
	Application Timeline Trace				
	Power Profiling				
	Select Power Profiling Counters				
	Pro	ofile Settings			

- 4. Select the desired process from the list of attachable processes and click the "Attach" button to start profiling.
- 5. Optional: Pause / Stop the data collection of the profiled application using the execution toolbar buttons: III. Stopping/pausing the profile session is optional. You can let the application run to its end and then the profiling session will automatically end. For long running applications, pausing allows you to control when the profiling data collection occurs so that you can match the stage in your application execution that you want to profile.
- 6. When the profiled application execution is over, CodeXL processes the data collected, and a profile session window is opened. See the <u>View and Analyze the Profile</u> <u>Session Data Analysis</u> section.

Attach to Process dialog

p4v.exe2336x64AMD\salgranaraptr_im.exe5544x86AMD\salgranaRuntimeBroker.exe3520x64AMD\salgranaSearchUI.exe5576x64AMD\salgranaShellExperienceHost.exe5724x64AMD\salgranasihost.exe4152x64AMD\salgranasippingTool.exe9140x64AMD\salgranasvchost.exe5696x64AMD\salgranauCMapi.exe7860x64AMD\salgranaWINWORD.EXE3164x64AMD\salgranaWINWORD.EXE8096x64AMD\salgrana	Process	ID	Туре	User Name	^
raptr_im.exe5544x86AMD\salgranaRuntimeBroker.exe3520x64AMD\salgranaSearchUl.exe5576x64AMD\salgranaShellExperienceHost.exe5724x64AMD\salgranasihost.exe4152x64AMD\salgranasizer.exe8508x86AMD\salgranaSnippingTool.exe9140x64AMD\salgranasvchost.exe5696x64AMD\salgranaucMapi.exe7860x64AMD\salgranaWINWORD.EXE3164x64AMD\salgranaWINWORD.EXE8096x64AMD\salgrana	OUTLOOK.EXE	3392	x64	AMD\salgrana	
RuntimeBroker.exe3520x64AMD\salgranaSearchUl.exe5576x64AMD\salgranaShellExperienceHost.exe5724x64AMD\salgranasihost.exe4152x64AMD\salgranasizer.exe8508x86AMD\salgranaSnippingTool.exe9140x64AMD\salgranasvchost.exe5696x64AMD\salgranataskhostw.exe3324x64AMD\salgranaUcMapi.exe7860x64AMD\salgranaWINWORD.EXE3164x64AMD\salgrana					_
SearchUI.exe5576x64AMD\salgranaShellExperienceHost.exe5724x64AMD\salgranasihost.exe4152x64AMD\salgranasizer.exe8508x86AMD\salgranaSnippingTool.exe9140x64AMD\salgranasvchost.exe5696x64AMD\salgranataskhostw.exe3324x64AMD\salgranaUcMapi.exe7860x64AMD\salgranaWINWORD.EXE3164x64AMD\salgrana					
ShellExperienceHost.exe5724x64AMD\salgranasihost.exe4152x64AMD\salgranasizer.exe8508x86AMD\salgranaSnippingTool.exe9140x64AMD\salgranasvchost.exe5696x64AMD\salgranataskhostw.exe3324x64AMD\salgranaUcMapi.exe7860x64AMD\salgranaWINWORD.EXE3164x64AMD\salgrana	RuntimeBroker.exe	3520	x64	AMD\salgrana	
sihost.exe 4152 x64 AMD\salgrana sizer.exe 8508 x86 AMD\salgrana SnippingTool.exe 9140 x64 AMD\salgrana svchost.exe 5696 x64 AMD\salgrana taskhostw.exe 3324 x64 AMD\salgrana UcMapi.exe 7860 x64 AMD\salgrana WINWORD.EXE 3164 x64 AMD\salgrana	SearchUI.exe	5576	x64	AMD\salgrana	
sizer.exe 8508 x86 AMD\salgrana SnippingTool.exe 9140 x64 AMD\salgrana svchost.exe 5696 x64 AMD\salgrana taskhostw.exe 3324 x64 AMD\salgrana UcMapi.exe 7860 x64 AMD\salgrana WINWORD.EXE 3164 x64 AMD\salgrana WINWORD.EXE 8096 x64 AMD\salgrana	ShellExperienceHost.exe	5724	х64	AMD\salgrana	
SnippingTool.exe9140x64AMD\salgranasvchost.exe5696x64AMD\salgranataskhostw.exe3324x64AMD\salgranaUcMapi.exe7860x64AMD\salgranaWINWORD.EXE3164x64AMD\salgranaWINWORD.EXE8096x64AMD\salgrana	sihost.exe	4152	х64	AMD\salgrana	
sychost.exe5696x64AMD\salgranataskhostw.exe3324x64AMD\salgranaUcMapi.exe7860x64AMD\salgranaWINWORD.EXE3164x64AMD\salgranaWINWORD.EXE8096x64AMD\salgrana	sizer.exe	8508	x86	AMD\salgrana	
taskhostw.exe3324x64AMD\salgranaUcMapi.exe7860x64AMD\salgranaWINWORD.EXE3164x64AMD\salgranaWINWORD.EXE8096x64AMD\salgrana	SnippingTool.exe	9140	x64	AMD\salgrana	
UcMapi.exe 7860 x64 AMD\salgrana WINWORD.EXE 3164 x64 AMD\salgrana WINWORD.EXE 8096 x64 AMD\salgrana	svchost.exe	5696	x64	AMD\salgrana	
WINWORD.EXE 3164 x64 AMD\salgrana WINWORD.EXE 8096 x64 AMD\salgrana	taskhostw.exe	3324	x64	AMD\salgrana	
WINWORD.EXE 8096 x64 AMD\salgrana	UcMapi.exe	7860	x64	AMD\salgrana	
	WINWORD.EXE	3164	x64	AMD\salgrana	
xgTraylcon.exe 6972 x86 AMD\salgrana 🗸	WINWORD.EXE	8096	х64	AMD\salgrana	
	xgTraylcon.exe	6972	x86	AMD\salgrana	~
	Show non-attachable process System-Wide profiling				

The dialog shows all the processes in the system, which are divided into 2 groups:

- 1. **Attachable processes:** These are the processes which the current user may attach to. Note that a user may not attach to other users' processes without having administrative privileges.
- 2. Non-attachable processes: These processes cannot be attached to due to certain privilege protections. These processes are not viewed by default, and are viewed only if the "Show non-attachable processes" checkbox is checked.

The list of processes is static, and may be refreshed by clicking the "Refresh" button (or reopening the dialog).

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Session > CPU Profile Global Settings

CPU Profile Global Settings

The CPU Profile global settings are set when the project is initially created. These settings affect every session and control how AMD CodeXL displays profile data.

To set the options:

- 1. In the CodeXL menu, click **CodeXL Options**. The Edit CodeXL Global Settings dialog box is displayed.
- Select the CPU Profile tab.
 The CPU Profile tab controls the display of source files and symbolic information. See the description for each of the global profile settings below.
- 3. Do one of the following:

Click **OK** to activate the new options and close the dialog box,

Click **Restore Default Settings** to reset the dialog box selections to the system default, or

Click **Cancel** to close without committing any changes.

<u>G</u> eneral	GPU <u>D</u> ebug	CPU Profile	Analyze				
Symbols							
Addi	itional debug sy	mbol paths					1
Syr	mbol Server Dire	ectories					
Down	load directory						
	New E	<u>R</u> emove		P	Nove Up	Move <u>D</u> own	
	http://msdl.mi	crosoft.com/d	lownload/symbols				
	nttp://msdl.mi	crosoft.com/d	lownload/symbols				
	nttp://msdl.mi	crosoft.com/d	lownload/symbols				

Additional Debug Symbol Paths	Select least one symbol server.
Symbol Server Directories	Select directories for the symbol server.
Block size in bytes to fetch when scrolling through disassembly	The size of the data block that is being automatically fetched when scrolling through disassembly.

 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Session > CPU Profile Project Options

CPU Profile Project Options

Use the CodeXL Project Settings dialog to configure the current project CPU profile settings:

General Settings Page

To configure a specific application for profiling, enter the path to the application executable in the 'Executable Path' field.

This field can be left empty if you intend to perform System Wide profiling.

To profile a Java application:

- Enter the path of the Java runtime executable in the 'Executable Path' field, e.g. "C:\Program Files\Java\jre7\bin\java.exe"
- Enter the path of the java classes top folder in the "Working Directory" field, e.g. "C:\Scimark2"
- Enter the name of the main java class in the command line arguments, e.g. "jnt.scimark2.commandline"

The screenshot below shows an example of configuring a Java application for profiling.

Profile Settings Page

Click File -> Project Settings and select the Profile tree node.

The "Profile Type" settings page will help you set the profile type of the current project and the profile scope:

CodeXL Project Settings		X
General Pro Debug Profile Pro	file Type ile Session Type: CPU: Time-based Sampling Identify the hot spots in a program that are consuming the most time ile Scope: O Single Application Profile (java.exe) O System-Wide Profile Image: System-Wide Profile with focus on application (java.exe)	X
Restore Default Settings		OK Cancel

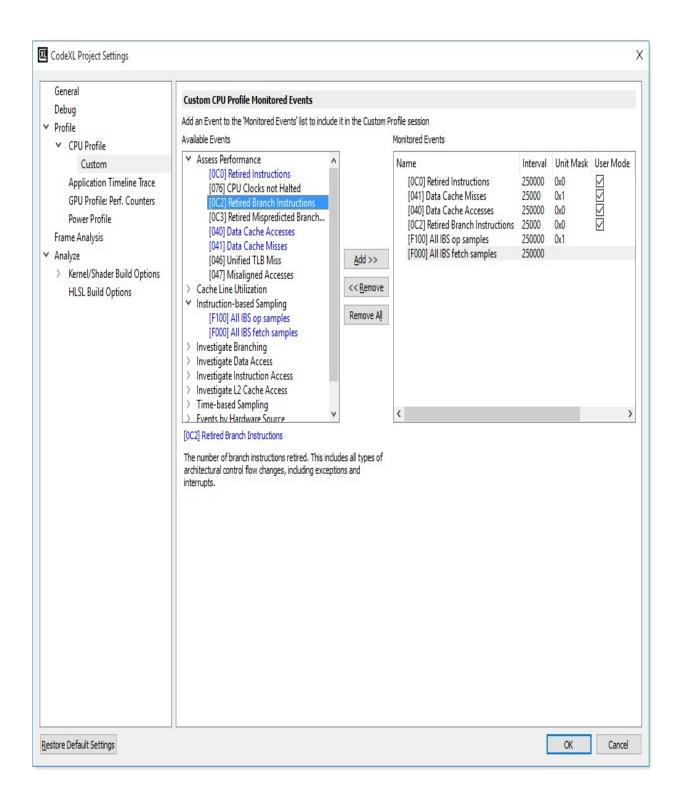
Displays the currently selected **<u>Profile Type.</u>**

Profile Session Type	The combo box contains both CPU and GPU profile types. Use this to select which type of data you want to collect while running the next profile session. Use the text below the profile type combo box to get a description of each of the profile types.
Profile Scope	Defines the scope that the next profile session will monitor. Single Application – the profile session will collect data for only the profiled application (defined in the "General" settings page). System-Wide Profile – the profile session will collect data for each of the running processes in the system.
	System-Wide Profile with focus on application – the profile session will collect data for each of the running processes in the system. The profile session will also collect call stack details for the focused application (defined in the "General" settings page).

CPU Profile Settings Page

Click Profile -> Profile Settings to open this settings page.

The CPU Profile settings page contains CPU profile specific configurations.



Checking this option will set CodeXL to

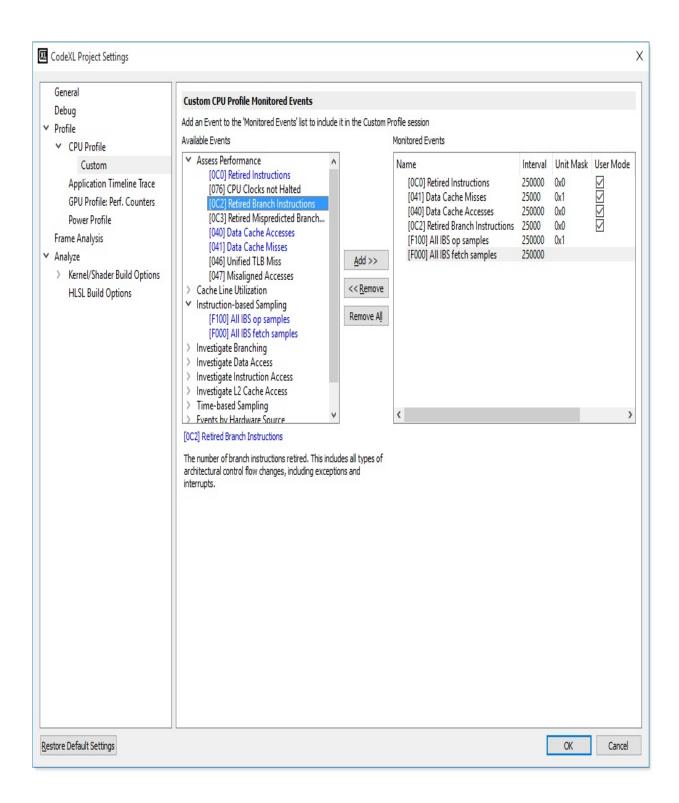
Collect call stack details	collect call stack details while profiling. Note : Call stack collection has higher overhead compared to sessions without call stack sampling, because stack unwinding is done whenever a sample is taken for the target process. The unwind operation, combined with the larger amount of data that must be written to the trace file, creates the higher overhead. Also, because of OS limits, the complete call stack might not be available Set the appropriate "Call stack collection depth" and "Call stack every" options, to balance between performance overhead and statistical accuracy.
Collect for code executed in	Limits the collection of call stacks only for code running in User space, Kernel space or both (User space and Kernel space).
Collect call stack every	This parameter will define the frequency of call stack collection.
Call Stack Collection Depth	Defines the level of depth for the collected call stack. A higher depth will require performance overhead, but the data of the collected call stack will be more accurate. Options are: Minimal / Low / Medium / High / Maximal.
	Selecting one of these options will sample the call stack with an up to the depth of 2 / 32 / 64 / 128 / 392 levels.
	Note: This option is set separately for Time- Based Sampling and other CPU Profiling session types.

Reproduce missing call stack info	 Perform additional analysis to overcome frame-pointer omission (FPO) in 32-bit apps and lack of unwind info in 64-bit. The profiler will store additional data during the profile session and require more time during postsession processing. Note: This option is set separately for Time-
	Based Sampling and other CPU Profiling session types.
	Configures the schedule of collecting data during the profile session execution.
	Throughout entire duration – collect data throughout entire profile session execution
Collection Schedule	Start profile with collection paused - the profile session will start with no data collection. Use the "Pause" button to resume data collection on the profiled application.
	Scheduled – set the specific timing for data collection during the execution of the profile session.
Start data collection after	Start data collection with a delay of 'X' seconds from the start of the profile session
End data collection after	When selected, the profile session data collection will end after 'X' seconds.
Then, terminate the process	When checked, after 'X' seconds selected to end the data collection after, the profiled process will be terminated by CodeXL.

Profile hardware scope	Use the tree structure of the existing hardware cores, or the affinity mask, to define the cores on which the profiled data will be collected.
Restore Default Settings	Restore the original default settings.

CPU Profile Custom Settings Page

Click Profile -> Profile Settings to open this settings page. Select "Profile -> CPU Profile -> Custom" tree node to go to the custom settings page.



This settings page configures the list of events collected while running a session of type "Custom Profile". Available Events lists all the available events.

Monitored Events lists the events which are selected for monitoring.

Use **Add**, **Remove**, **Remove All** buttons to add the selected event(s), remove the selected event(s), or remove all the selected events.

- 1. Some events have a checkbox:
 - Usr Enables the collection of user-level samples for an event.
 - **Os** Enables the collection of operating system-level samples for an event.
 - Edge Enables the edge- and level- detection that control the way an event signal is sensed; this affects the way an event is counted in a performance counter.

The available events depend on the CPU hardware in your system. Note that only one **Timer** event, **IBS all op** samples event, or **IBS fetch** sample event can be monitored at a time.

Available Events	Lists the available hardware events for profiling, nested within the configurations.
Add / Remove buttons	Adds to, or removes from the Monitored Events table the selected available event or configuration.
	A list containing the selected events for data collection in the "Custom Profile" sessions. A monitored event item in this list will contain: Name - The name of the monitored event.
	Interval – the period of how often a sample is to be taken (for counting occurring events).
	Unit Masks - Used to specify the unit mask setting for the selected event. Each bit set has a different meaning. The Event Settings pane gives you a description of the current setting.

Monitored Events	 Usr - Enables collection of user-level samples for an event. Os - Enables collection of operating system-level samples for an event. 						
	 Edge - Edge detection and level detection control the way an event signal is sensed; it affects the way an event is accumulated as a count in a performance counter. The occurrence of an event (a hardware condition) is asserted as a physical hardware signal. An event has a duration that can be as short as a single CPU clock cycle or it can be several cycles long. When OK is clicked and the Edge checkbox is 						
	checked, the "Custom Profile" configuration is selected automatically, and profiling can start immediately.						

Notable Available Events

[E000] Timer event	Hardware APIC timer event. The default is 1ms. The minimum interval is 0.1ms.
[F000] IBS fetch samples	Determines how often an IBS fetch sample is taken. IBS fetch sampling counts completed fetches to determine when the next IBS fetch sample is taken.
	Determines how often an IBS op sample is taken. When the Unit Mask is 0x0 (Count clock cycles), IBS op sampling counts processor cycles to determine

[F100] IBS all op samples	when the next IBS op sample is taken. When the Unit Mask is 1 (Count ops dispatched), IBS op sampling counts dispatched ops to determine when the next IBS op sample is taken. Dispatched op counting is the preferred mode because profiles produced through cycle counting can be biased by instruction timing.
Events by Hardware Source	Performance monitoring counter events that vary according to the system's hardware. The individual descriptions are displayed when the event is selected.
	A hardware event can be added multiple times, but the unit mask, Usr, or Os settings must be different. When an available event or configuration is selected, a description is shown below the list. Most event details can be directly edited within the table by clicking on the detail to change. When an event is selected, the unit mask details are also shown below the Monitored Events table.
	The presence of a hardware condition is asserted when the event signal is high. Absence of the condition is asserted when the event signal is low. When edge detection is used, each low-

HardwareEventNotes Hardware Performance Counter Notes

to-high transition of the event signal is counted as a single event; that is, the performance counter is incremented by one. When level detection is used, the level is sensed during each clock cycle, and the performance counter is incremented by one for each cycle during which the event signal is asserted.

The typical setting for an event is level-detect (the Edge box is unchecked). For example, if the event signal represents the "NOT HALTED" CPU state, then the performance counter counts the number of CPU clocks that the CPU spent in the "NOT HALTED" state. This configuration corresponds to the conventional "CPU Clocks Not Halted" event.

Performance counters count either specific processor events or the duration of events. The "Dispatch Stalls" counter event, for example, measures the number of processor cycles when the instruction decoder has stalled for any reason. Edge detection and level detection can be used to determine the average number of cycles per stall by:

 Counting the number of dispatch stall cycles (level detection).

	 Counting the number of dispatch stalls (edge detection). Dividing the number of stall cycles by the number of stalls.
--	--

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Data Analysis

CPU Profile Data Analysis

The data and source results of a profile session are displayed in the form of tabulated information and annotated source code.

Use the CodeXL Explorer to navigate between tabs of the same sessions, and between different sessions.

Profile session results contain several pages that can be accessed through the CodeXL Explorer and by using the context menus for any of the profile session data tables.

The following views can be displayed for each session:

CPU Profile Session Explorer Overview Page Profile Session Modules View Profile Session Functions View Profile Session Call Graph View Profile Session Source or Disassembly View Profile Session Display Settings Import a Profile Session Save CPU Profile Data

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Data Analysis > CPU Profile Session Explorer

CPU Profile Session Explorer

The CPU Profiling project consists of the CodeXL Explorer pane, the data pane, menu and toolbar commands, and subsequent tabs and views that are displayed during the profile sessions.

The CodeXL Explorer pane lists the profile sessions collected for the current project, organized by profile session type.

The CodeXL Explorer lets you:

- Double-click on a session node in the tree, to open it in the data pane.
- Right-click on a tree node to delete, rename, or import sessions.
- Right-click on a session tree node to browse the folder that contains the session. In the screenshot below you can see the CodeXL explorer for "classic" project. The session in bold is the currently selected profile session, and the profile type in bold font (CPU: Assess Performance) is the profile type for the currently selected session.

🖇 👌 🕶 🔟 🖉 🕨 Þ. Local 🔻 📗 📗										
deXL Explorer	đ×	🖄 Apr-10-2016_13-00-20 (CPU: Time-based Sampling) 🔟 🕺 Apr-10-2016_13-00-23 (CPU: Time-based Sampling) 🔟 🛕 Apr-10-2016_13-00-28 (CPU: Assess Performance) 🔯								
	X	0 Profile Overview								
 CodeXL Matrix Multiply Profile Mo OCPU: Time-based Sampling A 10 2016 12 00 20 	ode (CP	5 Hottest Functions Display: System Modules Hidden Hotspot Indicator Data Cache Accesses 💌								
 Ó Apr-10-2016_13-00-20 Ó Ó Overview 		Function	Sam	oles % of Hotspot Samples	Module					
Modules		📓 classic_multiply_matri	ces(void) 3,066	97.77%	AMDTClassicMath	Mul.exe				
E Call Graph		initialize_matrices(void		2.14%	AMDTClassicMath					
Functions		#4 BHDrvx64.sys!0xf80177	'13adf2 1	0.03%	BHDrvx64.sys					
✓ ₫ Apr-10-2016_13-00-23		HDrvx64.sys!0xf80177		0.03%	BHDrvx64.sys					
Overview		BHDrvx64.sys!0xf80177	'Ibaaбe 1	0.03%	BHDrvx64.sys					
Modules Call Graph Functions COU: Assess Performance										
✓ ▲ Apr-10-2016_13-00-28		5 Hottest Processes		5 Hottest Modules						
Overview		Process	PID Samples	% of Hotspot Samples		Module	Samples 5	% of Hotspot Samples		
🖗 Modules		AMDTClassicMatMul.exe	1884 3,468	96.07%		AMDTClassicMatMul.exe	3,133	99.90%		
 Call Graph Functions 		conhost.exe	7320 142	3.93%		₩ BHDrx64.sys	3 (),10%		
		٢			>	٢				
perties	đx	Execution	- In	1.501	n la juptel i	u u l				
U: Profile Session	٨			deXL\examples\ClassicMatMul\		Matmuliexe				
file Type Assess Performance				ents\CodeXL\examples\Classid		ultili pulling to the second	10.00.00			
sion Name Apr-10-2016_13-00-28		Profile Session Directory Command Line Argumen		na yappData (Koaming (AMD (Co	dext (CodeXL Matrix M	/ultiply_ProfilerOutput\Apr-10-2014	5_13-00-28			
cutable Path: C: Users Public Documents C	odeXL									_

Session Views Right-Click Menus

Another way to navigate between the profile session views is to use the session views' right-click menus. For example, in the screenshot below, the function "multply_matrices" is marked as a hot-spot in the overview page. Right-click on the function to see the following context menu commands:

Opening Source Code opens the source code for "multply_matrices" to show how the data samples are distributed within the selected function lines.

Display in Call Graph View opens the call graph view to show the "multply_matrices"' call paths.

Display in Functions View opens the functions view to see how the data is distributed within the functions in this module (classic.exe).

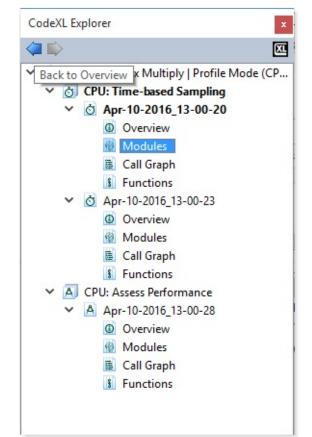
Function	Samples	% of Hotspot Samples	Module	
multiply_matrices(void)	4,081	95.00%	classic.exe	0 0 01
initialize_matrices(void)	58	1.35%	classic.exe	Open Source Code
getptd	36	0.84%	classic.exe	Display in Call Graph View
rand	26	0.61%	classic.exe	Display in Functions View
detLastError	16	0.37%	kernel32.dll	
other	79	1.84%		Сору
				Select All

After clicking the "Display in Call Graph" command, the call graph view is opened, with the function "multiply_matrices" selected. When right-clicking on the function, a context menu is opened with the option to either display the function in source code or in functions view.

Functions			D	isplay: <u>All Mod</u>	ules	Process: dassic.exe(492) 🔻 Monitored event: DC accesses			
Function (21 functions)	Self Samples	Deep Samples	% of Deep Samples	No. of Paths	Source File	Module		-	
RtlInitializeExceptionChain BaseThreadInitThunk		4275	99.98% 99.98%	13		ntdll.dll			
multiply_matrices(void)	4081	4275 4081	99.98% 95.44%	13 1	classic.cpp(59)	kernel32.dll classic.exe	Open Source Code	1	
initialize_matrices(void)	58	58	1.36%	1	classic.cpp(36)	classic.exe	Display in Functions View		
_getptd	36	58	1.36%	4	tidtable.c(289)	classic.exe	Сору		
rand	26	26	0.61%	1	rand.c(70)	classic.exe	Select All		
GetLastError	16	16	0.37%	1		kernel32.dll			
FIsGetValue	14	14	0.33%	2		KernelBase.dll			
SetLastError	12	12	0.28%	1		kernel32.dll			
FIsGetValue	9	9	0.21%	1		kernel32.dll		-	

CodeXL Explorer Navigation Arrows

The CodeXL Explorer arrows record the navigation to each of the session views that were opened. Use these arrows to go back and forward between the displayed session views.



 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Data Analysis > Profile Session Overview Page

Profile Session Overview Page

The CPU Profile Session Overview Page describes an overview of the data collected during the profile session.

The top section of the overview page contains the following data tables: Hot Spot Functions table, Hot Spot Modules table and Hot Spot Processes table. Note that the Processes table is only displayed when running a system-wide profile session.

The overview data tables can get a zoom-out image of the performance of the executed application. Use these tables to review the summary of hot spot locations in your application. The tables display the data organized by a 'hot spot indicator'.

The bottom section of the overview page contains general information for this session.

CodeXL Matrix Multiply - CodeXL Profile Mod Eile Edit View Debug Profile Frame Ana		- 0
Image: Second		🙆 Apr-10-2016_13-00-23 (CPU: Time-based Sampling) 🔟 🔺 Apr-10-2016_13-00-28 (CPU: Assess Performance) 🗵
CodeXL Matrix Multiply Profile Mode (CP CPU: Time-based Sampling Apr-10-2016_13-00-20	5 Hottest Functions	Display: System Modules Hidden Hotspot Indicator Data Cache Accesses
 Overview Modules Call Graph Functions Apr-10-2016_13-00-23 Overview Modules Call Graph 	Function Samples % of Hots classic_multiply_matrices(void) 3,066 97.77% initialize_matrices(void) 67 2.14% initialize_matrices(void) 67 2.14% initialize_matrices(void) 67 0.03% initialize_matrices(void) 67 0.03%	spot Samples Module AMDTClassicMatMul.exe AMDTClassicMatMul.exe BHDrxx64.sys BHDrxx64.sys BHDrxx64.sys
Functions CPU: Assess Performance A pr-10-2016_13-00-28	5 Hottest Processes	5 Hottest Modules
Overview Overview Modules Call Graph Functions	Process PID Samples % of Hotspot AMDTClassicMatMul.exe 1884 3,468 96.07% conhost.exe 7320 142 3,93%	t Samples Module Samples % of Hotspot Samples AMDTClassicMatMul.exe 3,133 99.90% Mathematical BHDrxx64.sys 3 0.10%
	٢	> <
perties 🗗 🛪		
U: Profile Session Assess Performance ssion Name Apr-10-2016_13-00-28 ecutable Path: C: (Users)Public/Documents/CodeXL amples/ClassidMatMul/Release IDTClassidMatMul.exe	Working Directory: C: Users Public Documents (CodeXL)ex Profile Session Directory: C: Users (salgrana) AppData (Roc Command Line Arguments:	
orking Directory: C:\Users\Public:\Documents		איז קאראיזואייהב שינופרובהם אי זיטאי קקאבמושה. סכובע מ שרוכרוגר ואפקשיע ברשוגמטי עי שוקאמי וזענגבע ושרגשה שמצע שר

Hot Spot Indicator

The screenshot above displays an Assess Performance session overview. The combo box at the right side of the view is a hotspot indicator combo box. Selecting another field in this combo box causes the data in the view to be re-calculated. The samples column contains the hot spot

functions/modules/processes calculated with the new indicator. The screenshot below displays the same session overview, after selecting "CPU clocks not halted (cycles)" in the hot spot indicator combo box. The tables are updated to display hottest functions / modules/ processes according to each of the monitored event / data field listed in the combo box. The table is updated with the CPU clocks sample count and percent, and the "Samples" column tooltip now specifies the CPU Clocks event.

Profile Overview			
5 Hottest Functions			Display: <u>All Modules</u> Hotspot Indicator CPU docks not halted (cycles)
Function	Samples	% of Hotspot Samples	Module
multiply_matrices(void)	66,071	Samples collected for the Hotspo	oot Indicator (CPU clocks not halted (cycles))
💒 KeUpdateRunTime	249	0.37%	ntoskrnl.exe
initialize_matrices(void)	61	0.09%	classic.exe
😹 _getptd	39	0.06%	classic.exe
ExEnterPriorityRegionAndAcquireResourceShared	27	0.04%	ntoskrnl.exe

Session Properties Section

The bottom section of the overview page contains the profiled session properties.

Execution							
Target Path: C: \CodeXL \classic.exe							
Working Directory: C:\CodeXL							
Data Folder: C: \Users \AMD \AppData \Roaming \AMD \CodeXL	\dassic_ProfilerOutput\Sep-02-2014_04-30-43						
Command Line Arguments:							
Environment Variables:							
CPU Affinity: 0x3f	Profile Scope: Single Application						
Call Stack Sampling: True Call Stack Information: Scope - User space, Depth - 32, Frequency - per 1 samples, FPO - Off							
Profile Details							
Profile Session Type: Assess Performance	Monitored Events						
Profile Start Time: Sep-02-2014_04-30-43	Data cache accesses	Data cache misses	Misaligned accesses				
Profile End Time: Sep-02-2014_04-30-54	CPU clocks not halted (cycles)	Retired instructions	Retired branch instructions				
Profile Duration: 10 seconds	Retired mispredicted branch instruc	ctions 1 DTI B and 2 DTI B r	nice				
CPU Details: Family 0x10, Model 10, 6 core(s)	Rearea hispreakted brand hisa av		135				
Total Processes in Profile: 1							
Total Processes III Profile; 1							

Execution	 Contains the session execution properties: Target Path, Working Folder, Data Folder, Command Line Arguments and Environment Variables. These session settings are general CodeXL project settings. Use the CodeXL Project Settings Dialog to configure them. CPU Affinity, Call Stack Sampling, System-Wide Profile, Call Stack Information -these settings are CPU profile project settings. Use the CodeXL CPU Profile Project Settings
Profile Details	Dialog to configure them. Contains the following information of the profile session: Profile session type (See CPU Profile Types for details), Profile start-end time, Duration, CPU Family and Total Amount of Processes and Threads. This section also contains a list of monitored events collected for this session.

Navigating to other views

The Overview page describes a zoomed-out image of the profiled session. For more information about bottle necks in your application, use **Modules View**, **Functions View**, and **Call Graph View**. You can also open a **Source-Code View** to see a source-line level performance data. These views can be opened using the **CodeXL Explorer Tree**. Each of the tables in the overview page implements a right-click menu that lets you navigate between session views.

Shoueser	unctions				UIS	piay: <u>All Module</u>	B Hotspot Indicator Da	ta cache accesses
Function			Samples	% of Hotspot Samples	Module			
👷 multiply	_matrices	(void)	975	64.10%	classic.exe			
a _getptd a∰ memcp	у		48 46 28 Data::Data *) 25 399	3.16% 3.02% 1.84% 1.64% 26.23%	classic.exe classic.exe win32k.sys Qt5Core.dll		ce Lode Call Graph View Functions View	
5 Hottest Pr	ocesses				5 Hottest Moo	dules		
Process	PID	Samples	% of Hotspot Samp	oles	Module	Samples	% of Hotspot Samples	
classic.exe	1660	1,188	76.35%		classic.ex	e 1,090	70.05%	
CodeXL.exe svchost.exe System Idle conhost.exe other	1016 0	198 52 36 25 57	Display in Modules Display in Functions Copy Select All		 ntoskrnl.e win32k.sy Qt5Gui.dl Qt5Core.c other 	rs 64 I 60	Display in Function Display in Modules Copy Select All	

Per-Process vs System-Wide Profile Session

A CPU Profile session can be:

- \cdot $\,$ Per-process Only the target application is profiled. This is the default.
- System-Wide Samples are taken for all running processes in the system. Use the <u>CodeXL CPU Profile</u> <u>Project Settings Dialog</u> to check / un-check the systemwide profile option.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Data Analysis > Profile Session Modules View

Profile Session Modules View

This displays a module-by-module detailing of performance data. Use this view to see the distribution of hot spots among the modules of your application. The data displayed in this view is collected for the session profile type. The screenshot below displays an Assess Performance profile session, with a single process (classic.exe). Modules with a high sample count usually indicate performance bottlenecks. Sort the modules table according to a specific counter to highlight potential bottlenecks.

eXL Explorer	🗗 X 👌 Ap	r-10-2016_13-00-20 (CPU: Time-ba	ased Sampling) 🗵 🧯	🙋 Apr-10-2016_13-00-23 (CPU: Time-based Sampling) 🙎 🛛 🔺 Apr-10-2016_13-00-28 (CPU: Assess Performance) 🛽					<u>)</u>
		ofile Overview							
 CodeXL Matrix Multiply Profile Mode OPU: Time-based Sampling 	(CP Funct	ions		Display: <u>All Dat</u>	a, All Modules	17 modules sh	own, 0 modules hidden Proc	cess All Processes	
Y 👌 Apr-10-2016_13-00-20	Func	tion		Module	Timer				
Overview		HDrvx64.sys!0xf80177138f21		BHDrvx64.sys	1				
🔞 Modules		lassic_multiply_matrices(void)		AMDTClassicMatMul.exe	490				
📱 Call Graph	and the second se	ConsoleCreateloThread		ConhostV2.dll	1				
§ Functions		EC_SHARE_REF_CNT		win32kbase.sys	1				
 Apr-10-2016_13-00-23 		nterCrit		win32kbase.sys	1				
0 Overview		xAcquirePushLockExclusiveEx		ntoskrnl.exe	2				
Modules		xAcquireResourceExclusiveLite		ntoskrnl.exe	1				
📓 Call Graph		xAcquireRundownProtectionCa	cheAwareEx	ntoskrnl.exe	1				
§ Functions		xAllocatePoolWithOuotaTag		ntoskrol eve	1				
 A CPU: Assess Performance 		xAllocatePoolWithTag	ExAcquireRundownPro	ectionCacheAwareEx	1				
 Apr-10-2016_13-00-28 		xFreePoolWithTag		ntoskrnl.exe	1				
Overview	₩ ⁶⁴ E	xReleaseCacheAwarePushLockSI	haredEx	ntoskrnl.exe	2				
Modules	₩ ⁴ E	xReleasePushLockExclusiveEx		ntoskrnl.exe	2				
E Call Graph	F F	irstHookFunc		sysfer.dll	1				
Functions	F F	lsGetValue		KernelBase.dll	2				
a runctions	₩ ⁶⁴ F	ltQueryInformationFile		FLTMGR.SYS	1				
	₩ ⁶⁴ F	sRtIAcquireHeaderMutex		ntoskrnl.exe	1				
	🔣 G	etAdjustObjectAttributesForPriv	ateNamespaceRoutine	KernelBase.dll	1				
	🔣 G	ietLastError		KernelBase.dll	4				
	💑 G	ietMonitorInfoW		user32.dll	1				
	in the second	lalFlushCommonBuffer		hal.dll	6				
		bmSelectBitmap		win32kbase.sys	1				
		ImgLockAllOwners		win32kbase.sys	1				
		nitialize_matrices(void)		AMDTClassicMatMul.exe	1				
		nitialize_narrow_environment		ucrtbase.dll	2				
		oGetStackLimits		ntoskrnl.exe	1				
		eReleaseInStackQueuedSpinLoc		ntoskrnl.exe	1				
		eSetEvent		ntoskrnl.exe	3				
	₩ ⁴ K	eSetPriorityThread		ntoskrnl.exe	1				

The modules view displays two tables: processes and modules. The following screenshot shows the "classic" project in a system-wide session. The combo-box on the right side of the view lets you select display processes or modules in the top table.

Display Modules View by Processes

The top table displays processes. Select one or more processes in the top table, and the bottom table displays only the modules used by the selected processes.

Processes								Display: All Data, All Mod	dules Display by: Process	es
Process	PID	DC accesses	DC misses	Misalign access	CPU clocks	Ret inst	Ret branch	Ret misp branch	DTLB L1M L2M	
classic.exe	1660	1,188	1,236		16,139	2,000	1,400	10	3,803	
CodeXL.exe	688	.98	32	17	536	362	695	39	5	
svchost.exe	1016	52	8		233	142	277	5		
System Idle	0	36	39		2,702	63	97	34		
conhost.exe	2392	25	5		134	90	161	15	1	
System	4	20	4		121	55	97	21		
Explorer.EXE	1768	14	5		61	29	56	4		
svchost.exe	2964	13	2	1	51	42	101	3		
csrss.exe	500	3	5		99	25	36	9	1	
taskhost.exe	1276	L			2		1			
SearchIndexer.exe	2572	l			1		1			
Dwm.exe	232				1					
							1			
	-	v	ar DC mier	er Misalian acc	ors CDIL clocks	Retine	1	-h Ret micn brand	-h DTLR I 1 M I 2 M	
Module	Symbols	DC accesse		es Misalign acco						
Modules filtered I Module classic.exe	Symbols Loaded	DC accesse 1,090	es DC miss 1,227	es Misalign acco	15,899	1,812	906	ch Ret misp brand 8	:h DTLB L1M L2M 3,802	
Module classic.exe ntdll.dll	Symbols Loaded Not Loade	DC accesse 1,090 d 32		es Misalign acci	15,899 25	1,812 22	906 16			
Module classic.exe ntdll.dll KernelBase.dl	Symbols Loaded Not Loade	DC accesse 1,090 d 32 d 26		es Misalign acco	15,899 25 42	1,812 22 60	906 16 238			
Module classic.exe ntdll.dll KernelBase.dl kernel32.dll	Symbols Loaded Not Loade I Not Loade Not Loade	DC accesse 1,090 d 32 d 26 d 25	1,227	es Misalign acco	15,899 25 42 38	1,812 22 60 87	906 16 238 195	8	3,802	
Module classic.exe ntdll.dll KernelBase.dl kernel32.dll	Symbols Loaded Not Loade I Not Loade Not Loade Not Loade	DC accesse 1,090 d 32 d 26 d 25 d 13		es Misalign acci	15,899 25 42 38 125	1,812 22 60	906 16 238			
Module classic.exe ntdll.dll KernelBase.dl kernel32.dll fntoskrnl.exe	Symbols Loaded Not Loade I Not Loade Not Loade Not Loade Not Loade	DC accesse 1,090 d 32 d 26 d 25 d 13 d 1	1,227	es Misalign acci	15,899 25 42 38 125 3	1,812 22 60 87	906 16 238 195	8	3,802	
Module classic.exe ntdll.dll KernelBase.dl kernel32.dll fntoskrnl.exe win32k.sys fuafv.sys	Symbols Loaded Not Loade I Not Loade Not Loade Not Loade Not Loade	DC accesse 1,090 d 32 d 26 d 25 d 13 d 1 d 1	1,227 8	es Misalign acci	15,899 25 42 38 125 3 1	1,812 22 60 87 17	906 16 238 195 38	8	3,802	
Module classic.exe ntdll.dll kernelBase.dl kernel32.dll htskrnl.exe win32k.sys luafv.sys hts.sys	Symbols Loaded Not Loade Not Loade Not Loade Not Loade Not Loade Not Loade	DC accesse 1,090 d 32 d 26 d 25 d 13 d 1 d 1 d	1,227	es Misalign acci	15,899 25 42 38 125 3	1,812 22 60 87	906 16 238 195	8	3,802	
Module classic.exe ntdll.dll KernelBase.dl kernel32.dll htoskrnl.exe win32k.sys i luafv.sys	Symbols Loaded Not Loade Not Loade Not Loade Not Loade Not Loade Not Loade Not Loade	DC accesse 1,090 d 32 d 26 d 25 d 13 d 1 d 1 d 1 d 1 d	1,227 8	es Misalign acci	15,899 25 42 38 125 3 1 2	1,812 22 60 87 17	906 16 238 195 38	8	3,802	

Display by Modules

The top table displays modules. Select one or more modules in the top table, and the bottom table displays only the processes used by the selected modules.

/lodule			Symbols	DC acc	esses DC mis	ses Misalign	access	CPU clocks	Ret inst	Ret branch	Ret misp branch	
classic.e	(e		Loaded	1,090	1,227			15,899	1,812	906	8	
⁶⁴ ntoskrnl			Not Loaded	105	58			3,143	205	366	56	
win32k.s			Not Loaded	64	8	16			150	251	13	
Qt5Gui.c	-		Loaded	60	5	1	1	152	100	157	10	
Qt5Core	dll		Loaded	53	9		1	133	108	215	8	
ntdll.dll			Not Loaded	39	2		L	14	41	49		
KernelBa	se.dll		Not Loaded	26			L	12	60	239		
kernel32	dll		Not Loaded	25			3	39	89	195		
	n module	pid (1016)	Not Loaded	24	3		8	38	85	183	4	
Qt5Widg	ets.dll		Loaded	18	6		5	51	12	51	7	
		pid (2964)	Not Loaded	10	1	1	3	33	36	88	3	
fitmara	e	-4.	Not Loaded	5			1	1	A	10	5	
rocesses fil	PID	DC access		sses	Misalign access	CPU clocks	Ret inst	Ret branch	Ret m	isp branch	DTLB L1M L2M	
assic.exe	1660	1,188	1,236		-	16,139	2,000	1,400	10		3,803	
odeXL.exe	688	198	32		17	536	362	695	39		5	

Modules View Context Menus

The following two screenshots display the context menus for the modules and processes tables. The context menus lets you:

- Display in Functions View Displays the selected module
 / process in the functions view.
- Select "ModuleName.dll" in modules table Switches to

"Display by Modules" and selects ModuleName.dll in the modules table.

Module	DC accesses	DC misses	Misalign access	CPU clocks
⁶⁴ Explorer.EXE	6			54
Earth GROO∖	Display in Funct Display in Modu Select GROOVER	Iles View	dules Table	
	Copy Select All			

Process	PID	DC accesses	DC misses	Misalign access	CPU clocks	Ret i
ProtectionUt	4580	10	6		60	20
Rtvscan.exe	2492	2	2		18	4
SearchFilter	8	Disalauta				
SearchIndexe	2	Display in	Functions V	lew	26	16
SearchProto	4	Сору			2	
Smc.exe	1	Select All			404	128
SmcGui.exe	4	Select All			112	10
SynTPEnh.exe	4180					
System	4	1176	288	4	6462	2568
System Idle	0	462	332	2	17812	556
VolDanlu eve	5020					

Modules View Display Settings

Use the display filter link to open the **Display Settings Dialog** and change the properties of the currently displayed session.

Profile Overview Modules Display: All Data, All Modules Display by: Processes									
Process	PID	DC accesses	DC misses	Misalign access	CPU clocks	Ret inst	Ret branch	Ret misp branch	DTLB L1M L2M
classic.exe	1660	1,188	1,236		16,139	2,000	1,400	10	3,803
CodeXL.exe	688	198	32	17	536	362	695	39	5
svchost.exe	1016	52	8		233	142	277	5	

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Data Analysis > Profile Session Functions View

Profile Session Functions View

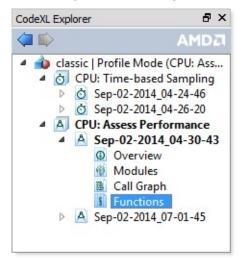
Function view displays list of functions called during profiling of the current session in a table.

odeXL Explorer		Apr-10-2016_13-00-2	3 (CPU: Time-based Sampling) 🗵 🛛 🔺 Apr-10-2016_13-00-28 (CPU: Ar	ssess Performance) 🗵
	🔟 🖉 Profile Overview 🔋 Functions 🗵			
 CodeXL Matrix Multiply Profile Mode OCU: Time-based Sampling 	(CP Functions	Display: <u>Al</u>	Data, All Modules 17 modules shown, 0 modules hidden Process All P	rocesses
✓ ₫ Apr-10-2016_13-00-20	Function	Module	Timer	
Overview	64 BHDrvx64.sysI0xf80177138f21	BHDrvx64.sys	1	
Modules	classic_multiply_matrices(void)	AMDTClassicMatMul	exe 490	
📕 Call Graph	ConsoleCreateloThread	ConhostV2.dll	1	
Functions	4 DEC_SHARE_REF_CNT	win32kbase.sys	1	
 Apr-10-2016_13-00-23 	EnterCrit	win32kbase.sys	1	
Overview	ExAcquirePushLockExclusiveEx	ntoskrnl.exe	2	
🖗 Modules	K ExAcquireResourceExclusiveLite	ntoskrnl.exe	1	
Call Graph	♣ ExAcquireRundownProtectionCacheAwareEx	ntoskrnl.exe	1	
§ Functions	* ExAllocatePoolWithOuotaTag	ntoskrol eve	1	
 CPU: Assess Performance 		lownProtectionCacheAwareEx	1	
✓ A Apr-10-2016_13-00-28	ExFreePoolWithTag	ntoskrnl.exe	1	
0 Overview	ExReleaseCacheAwarePushLockSharedEx	ntoskrnl.exe	2	
Modules	ExReleasePushLockExclusiveEx	ntoskrnl.exe	2	
E Call Graph	📓 FirstHookFunc	sysfer.dll	1	
Functions	🙀 FlsGetValue	KernelBase.dll	2	
a runctions	FltQueryInformationFile	FLTMGR.SYS	1	
	♣ ⁴ FsRtIAcquireHeaderMutex	ntoskrnl.exe	1	
	📓 GetAdjustObjectAttributesForPrivateNamespace	Routine KernelBase.dll	1	
	🙀 GetLastError	KernelBase.dll	4	
		user32.dll	1	
	HalFlushCommonBuffer	hal.dll	6	
	hbmSelectBitmap	win32kbase.sys	1	
	HmgLockAllOwners	win32kbase.sys	1	
	initialize_matrices(void)	AMDTClassicMatMul	exe 1	
	initialize_narrow_environment	ucrtbase.dll	2	
	IoGetStackLimits	ntoskrnl.exe	1	
	KeReleaseInStackQueuedSpinLock	ntoskrnl.exe	1	
	KeSetEvent	ntoskrnl.exe	3	
		ntoskrnl.exe	1	

Open Functions View

This view can be opened in any of the following ways.

 Double-click the Functions node of a session. It opens the functions view with Process selected as All Processes. If the view was already there, it just opens the view tab.



- Double-click the process table in the Profile Session Overview Page. This opens the Function view selecting that process. All the functions from that process are listed. This way of opening the Function view is possible only when multiple processes exist.
- Double-click the Module table in Profile Session Overview Page. This opens the Function view selecting only that module.
- Double-click the process table in Profile Session Modules View. This opens the Function view selecting that process.
- Double-click the Module table in Profile Session Modules View. This opens the Function view selecting only that module.
- Selecting the "Display in Functions View" option from context menu of any of the four tables of the Call Graph View. This opens the Function view selecting the function that was selected in Call Graph view on the current table.

Columns

This table shows two columns:

 Function – Contains the name of the function, if available, or a NO_SYMBOL string if the function name could not be determined from the available symbol. There is an icon before each function name in this column indicating if the function is a system function or a function from a user module. It also indicates if the corresponding library is 32bit or 64-bit.

Module – This column lists the corresponding module of the function. It lists the module name only, without path. Full path information is visible on the tooltips of the corresponding module.

Other optional columns are displayed only if the corresponding events are profiled and those columns are selected for display. For TBP, the only available variable column is timer. Multiple views are possible for EBP or IBS.

Each of the views has a corresponding list of selectable columns for display in the functions view. The combo-box at the beginning of **Column** section of **Display Settings** dialog lists the available views for the selected profile type of the current session.

Below the view combo-box is a list of checkboxes, each of which controls the visibility of a certain column. When the combo-box is checked, the corresponding column is visible; when the combo-box is unchecked, the corresponding column is hidden (see **CodeXL CPU Profile Display Settings Dialog**). For a full list of views and available columns, see the **CodeXL CPU Profile Types** section.

Filters

Functions table data can be filtered with one of the three options.

• **Display Settings dialog** – This is opened from the hyperlink above the table, labeled **Display**. This dialog lets you select the view, the displayed column(s) from them. It also let you specify whether to display system modules,

split the data based on core or NUMA nodes(splitting data based on NUMA nodes is available on Windows only), and display the absolute value of data or some percentage with a bar. (For details, see the <u>CodeXL CPU Profile Display</u> <u>Settings Dialog</u>.)

Functions				ta, All Modules <u>19 m</u>	odules shown, 0 mo	dules hidden Pro	cess classic.exe(4	92)
Function	Module	DC accesses	DC misses	Misalign access	CPU clocks	Ret inst	Ret branch	
multiply_matrices(void)	classic.exe	4,081	5,170		66,071	7,036	1,958	
initialize_matrices(void)	classic.exe	58			61	131	224	
getptd	classic.exe	36			39	53	170	
rand	classic.exe	26			20	23	52	
🔬 GetLastError	kernel32.dll	16			12	18	66	

Module Filter – This is opened from the hyperlink above the table. It shows the number of visible and hidden modules. Clicking on this hyperlink opens the **Module** Filter dialog. It contains a table with two columns: Modules and Path. Modules column lists the modules name (the icons have the same meaning as described in the function table), and Path contains the path of that module. Both columns are sortable by clicking on the respective heading.

Functions		Display: <u>All Dat</u>	ta, All Modules 19 m	odules shown, 0 mo	dules hidden Pro	classic.exe(49	
Function	Module	DC accesses	DC misses	Misalign access	CPU clocks	Ret inst	Ret branch
multiply_matrices(void)	classic.exe	4,081	5,170		66,071	7,036	1,958
initialize_matrices(void)	classic.exe	58			61	131	224
🙀 _getptd	classic.exe	36			39	53	170
🕷 rand	classic.exe	26			20	23	52
👷 GetLastError	kernel32.dll	16			12	18	66

The following screenshot shows an example result after clicking the hyperlink shown in the red boundary above.

🕰 CodeXL Modules Filter		×
Modules of System-wide Profiling(2 P	rocesses):	
Modules	Path	^
✓	C:\ProgramData\Symantec\Symantec Endpoint Protection\12.1.6318 C:\Users\Public\Documents\CodeXL\Examples\ClassicMatMul\Relea C:\Windows\SysWOW64\ C:\Windows\SysWOW64\ C:\Windows\SysWOW64\ C:\Windows\SysWOW64\ C:\Windows\System32\drivers\ C:\Windows\System32\Drivers\	
 ✓ [№] hal.dll ✓ [№] ntoskrnl.exe ✓ [№] ConhostV2.dll ✓ [№] NTES pyr 	C:\Windows\system32\ C:\Windows\system32\ C:\Windows\System32\ C:\Windows\System32\Drivers\	~
 ✓ Display System modules in Module ✓ Select all modules 		

Modules	This column contains one checkbox per cell which indicates whether to display functions of this module in functions view. There are two checkboxes below the table. The Select all modules checkbox is used to select/un- select all the modules currently displayed in the table. This checkbox is always enabled. The Display System modules in Modules Filter checkbox is enabled only if the Display system modules checkbox is checked (in the Display Settings dialog). When enabled, it shows / hides the system modules in the table. On selecting some modules in this table with the checkboxes and clicking the OK button, the functions view is updated. It shows the function only from selected modules, and the link above the table shows the updated count of shown and hidden
	the updated count of shown and hidden
	modules.

Processes	Process combo-box lists all the processes in process-name (process-id) format. It also has one option called All Processes . If a process is selected in this combo-box, all the
	functions from that process are listed in the table. If All Processes is selected, all the functions from all the modules of the current session are displayed in the current table.

unctions	I	Display: <u>All Data, All Modules</u>	84 modules sho	own, 0 modules hidden P	Process All Process	25
unction	Module	DC accesses	DC misses	Misalign access	CPU clocks	Ret inst
multiply_matrices(void)	classic.exe	975	1,227		15,782	1,689
initialize_matrices(void)	classic.exe	48			62	58
_getptd	classic.exe	46			36	50
🚰 memcpy	win32k.sys	28	4	16	64	46
QListData::dispose(struct QListData::Data *)	Qt5Core.dll	25	1		35	68

Context Menu

The Function table provides a context menu. Open it by right clicking on the table. The context menu has four items separated into two groups. The following list describes each of the four items.

unctions				Display: A	ll Data, All Modules	19 modules	s shown, 0 modules l	hidden Process classic.ex	(e(492)
Function	Module	DC accesses	DC misses	Misalign access	CPU clocks	Ret inst	Ret branch	Ret misp branch	DT
multiply_matrices(void)	classic.exe	4,081	5,170		66,071	7,036	1,958	26	15
initialize_matrices(void)	classic.exe	58			- 61	131	224		
getptd	classic.exe	36	(Open Source Code		53	170	1	
k rand	classic.exe	26	[Display in Call Graph Viev	v	23	52		
GetLastError	kernel32.dll	16				18	66		
FlsGetValue	KernelBase.dll	14	(Сору		22	204		
SetLastError	kernel32.dll	12	5	Select All		9	47		
FlsGetValue	kernel32.dll	9			19	12	48		
GetLastError	KernelBase.dll	8			9	9	54		
RtlRestoreLastWin32Error	ntdll.dll	8			16	13	24		

• Opening Source Code – Opens the source code or

disassembly view for the currently selected function in the function table.

- Display in Call Graph View Opens the Call Graph view for the currently selected function in the function table.
- Copy Copies the selected column(s) in the function table, along with the headers of the column in buffer, for pasting into another application such as Notepad, Microsoft word, Microsoft Excel, etc. See the common context menu section.
- Select All Selects all the rows and all the columns of the function table. See the common context menu section.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Data Analysis > Profile Session Call Graph View

Profile Session Call Graph View

The Call Graph view displays a list of functions with their Call Graph information, including caller-to-called relationship. This can be enabled or disabled from the Call stack collection checkbox of the CPU Profile Project Options for a C++ based session. Call Graph is not supported for a CLR- or Java-based session.

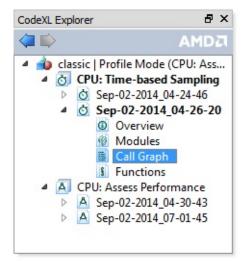
iodeXL Explorer 6	X 👌 Apr-10-2016_13-00-20 (CPU: Time-bas	sed Sampling) 🗵 (Apr-10-2016_13	-00-23 (CPU: Time-t	ased Sampling) 🗵	Apr-10-2	016_13-00-28 (0	CPU: Assess Performance) 🗵	
	🔟 🗕 Profile Overview 🔞 Modules 🗵	🎚 Call Graph 🗵							
🖌 👜 CodeXL Matrix Multiply Profile Mode (C	p								
Y 👌 CPU: Time-based Sampling	Functions		Display: All N	1odules	Process:	AMDTClassicMat	Mul.exe(3976)	 Monitored event: 	3
✓ Ø Apr-10-2016_13-00-20	Function (34 functions)	Self Samples	Deen Samples	% of Deen Samn	es No. of Paths	Source File		Module	-
Overview	_scrt_common_main_seh	och oumpres	505	96.74%		exe_common.ii	J(210)	AMDTClassicMatMul.exe	
10 Modules	main		498	90.74% 95.40%	-	amdtclassicma		AMDTClassicMatMul.exe	
Call Graph	classic_multiply_matrices(void)	496	496	95.02%	and the state of t	amdtclassicma	and the second	AN C:\Users\Public\Docum	-
 Functions Apr-10-2016_13-00-23 Overview 	initialize_matrices(void)	2	22	4.21%		amdtclassicma		AMDTClassicMatMul.exe	ICI
	rand	2	19	3.64%	8		11. 7	ucrtbase.dll	
Modules	qsort_s	8	17	3.26%	7			ucrtbase.dll	
E Call Graph	RtlSetCurrentTransaction		14	2.68%	5			ntdll.dll	
§ Functions	BaseThreadInitThunk		14	2.68%	5			kernel32.dll	
✓ ▲ CPU: Assess Performance	set_new_mode		13	2.49%	5			ucrtbase.dll	
✓ ▲ Apr-10-2016_13-00-28	wcsstr	6	6	1.15%	2			ucrtbase.dll	
 Modules Call Graph 	Immediate Parents and Children of Function Parents Sample	n: main s % of samples Mod	ule	Self	+ children	Samples	% of samples	Module	
§ Functions	_scrt_common_main_seh 498		TClassicMatMul.		lassic_multiply_ma		99.60%	AMDTClassicMatMul.ex	xe
		10010010			nitialize_matrices(v		0.40%	AMDTClassicMatMul.ex	
					self]	0		AMDTClassicMatMul.ex	.(e
	Paths			_ Sh	ow Call Graph selecti	on path			
	Function	Self sample	5 Downstream s	amples % of Do	wnstream samples	Module			-
	✓ RtlSetCurrentTransaction		7	1.41%		ntdll.dll			
	✓ RtlSetCurrentTransaction		7	1.41%		ntdll.dll			
	✓ BaseThreadInitThunk		7	1.41%		kernel32.dll			
	✓ _scrt_common_main_s	eh	7	1.41%		AMDTClassic	MatMul.exe		
	> main		7	1.41%		AMDTClassic			
	✓ scrt common main seh		491	98.59%		AMDTClassic	MatMul.exe		

Open Call Graph View

Open this view in one of the following ways.

B

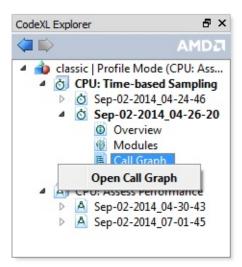
• Double click the **Call Graph** node of a session. If the Call Graph view was already there, this opens the Call Graph view tab.



Click the context menu item **Display in Call Graph View** of the functions table in the Profile Session Overview.

5 Hottest Functions					Display: All Modules Hotspot Indicator Timer
Function	Samples	% of Hotspot Samples	Module		
multiply_matrices(void	10,150	99.77%	classic.exe	Open Source Code	
🖈 _getptd	6	0.06%	classic.exe		
initialize_matrices(void)	4	0.04%	classic.exe	Display in Call Graph View	
FlsGetValue	4	0.04%	KernelBase.dll	Display in Functions View	
SetLastError	3	0.03%	kernel32.dll		
other	6	0.06%		Сору	

Click the context menu item **Open Call Graph** of **Call Graph** node of a session Page. This opens the Call Graph view with that process selected. All the functions from that process are listed. This way of invoking Function view is possible only when multiple processes exist.



Click the context menu item **Display in Call Graph View** of the functions table in the Profile Session Function View.

Function Mo	adula				
	odule	Timer		v	
multiply_matrices(void) class		10,150 6	Open Source Code		
initialize_matrices(void) clas	issic.exe		Display in Call Graph View		
SetLastError ker	rnelBase.dll rnel32.dll rnel32.dll	3	Copy Select All		

Tables in Call Graph View

There are four tables in the Call Graph View.

Function Table – Appears at the top of the page and contains the list of all functions (of the selected process or all processes if All Processes option is selected).

Functions				Display:	Il Modules	Process: dassic.exe(3020) Monitored event:
Function (13 functions)	Self Samples	Deep Samples	% of Deep Samples	No. of Paths	Source File	Module
RtlInitializeExceptionChain		10172	100.00%	8		ntdli.dli
BaseThreadInitThunk		10172	100.00%	8		kernel32.dll
mainCRTStartup		10164	99.92%	4	crt0.c(135)	classic.exe
main		10160	99.88%	3	classic.cpp(87)	classic.exe
multiply_matrices(void)	10150	10150	99.78%	1	classic.cpp(59)	classic.exe
initialize_matrices(void)	4	10	0.10%	2	classic.cpp(36)	classic.exe
_getptd	6	10	0.10%	2	tidtable.c(289)	classic.exe
rand		6	0.06%	1	rand.c(70)	classic.exe
FlsGetValue	4	4	0.04%	1		KernelBase.dll
SetLastError	3	3	0.03%	1		kernel32.dll

This table has the following columns.

FunctionFunction name that appeared in the call stack when the chosen monitored event triggered a sample collection. Based on the configuration of Display system modules in the Display Settings dialog, this includes the function from system modules.SelfThis shows the number of times this function was on the top of call stack when the chosen monitored event triggered a sample collection.Deep SamplesNumber of times this function was present in the call stack when the chosen monitored event triggered a sample collection.Mot of Deep SamplesPercentage of deep samples out of the total count of the samples gathered 100*(Deep Sample)/(Total Sample collected)No. of PathsNumber of unique paths containing this function.Source FileThe source files containing the function. The line number is shown in parentheses.ModuleThe name of the module that contains this function. Full path is available on tooltips text of the corresponding cell of the table.		
Functionconfiguration of Display system modules in the Display Settings dialog, this includes the function from system modules.SelfThis shows the number of times this function was on the top of call stack when the chosen monitored event triggered a sample collection.Deep SamplesNumber of times this function was present in the call stack when the chosen monitored event triggered a sample collection.Mot of Deep SamplesPercentage of deep samples out of the total count of the samples gathered 100*(Deep Sample)/(Total Sample collected)No. of PathsNumber of unique paths containing this function.Source FileThe source files containing the function. The line number is shown in parentheses.ModuleThe name of the module that contains this		stack when the chosen monitored event
Configuration of Display system modulesin the Display Settings dialog, thisin the Display Settings dialog, thisincludes the function from system modules.SelfThis shows the number of times thisSamplesfunction was on the top of call stack whenbeepNumber of times this function was presentSamplesNumber of times this function was presentbeepNumber of times this function was presentSamplesPercentage of deep sample collection.% of DeepPercentage of deep samples out of the total% of Deep100*(Deep Sample)/(Total SampleSamplesNumber of unique paths containing this function.No. of PathsNumber of unique paths containing this function.Source FileThe source files containing the function. The line number is shown in parentheses.ModuleThe name of the module that contains this function. Full path is available on tooltips	Function	55 1
includes the function from system modules.SelfThis shows the number of times thisSamplesfunction was on the top of call stack when the chosen monitored event triggered a sample collection.Deep SamplesNumber of times this function was present in the call stack when the chosen monitored event triggered a sample collection.% of Deep SamplesPercentage of deep samples out of the total count of the samples gathered 100*(Deep Sample)/(Total Sample collected)No. of PathsNumber of unique paths containing this function.Source FileThe source files containing the function. The line number is shown in parentheses.ModuleThe name of the module that contains this	1 unction	configuration of Display system modules
SelfThis shows the number of times this function was on the top of call stack when the chosen monitored event triggered a sample collection.Deep SamplesNumber of times this function was present in the call stack when the chosen monitored event triggered a sample collection.% of Deep SamplesPercentage of deep samples out of the total count of the samples gathered 100*(Deep Sample)/(Total Sample collected)No. of PathsNumber of unique paths containing this function.Source FileThe source files containing the function. The line number is shown in parentheses.ModuleThe name of the module that contains this function. Full path is available on tooltips		in the Display Settings dialog, this
Self Samplesfunction was on the top of call stack when the chosen monitored event triggered a sample collection.Deep SamplesNumber of times this function was present in the call stack when the chosen monitored event triggered a sample collection.% of Deep SamplesPercentage of deep samples out of the total count of the samples gathered 100*(Deep Sample)/(Total Sample collected)No. of PathsNumber of unique paths containing this function.Source FileThe source files containing the function. The line number is shown in parentheses.ModuleThe name of the module that contains this function. Full path is available on tooltips		5
Samplesthe chosen monitored event triggered a sample collection.Deep SamplesNumber of times this function was present in the call stack when the chosen monitored event triggered a sample collection.% of Deep SamplesPercentage of deep samples out of the total count of the samples gathered 100*(Deep Sample)/(Total Sample collected)No. of PathsNumber of unique paths containing this function.Source FileThe source files containing the function. The line number is shown in parentheses.ModuleThe name of the module that contains this function. Full path is available on tooltips		This shows the number of times this
Image: Note of the second se	Self	function was on the top of call stack when
Deep SamplesNumber of times this function was present in the call stack when the chosen monitored event triggered a sample collection.% of Deep SamplesPercentage of deep samples out of the total count of the samples gathered 100*(Deep Sample)/(Total Sample collected)No. of PathsNumber of unique paths containing this function.Source FileThe source files containing the function. The line number is shown in parentheses.ModuleThe name of the module that contains this function. Full path is available on tooltips	Samples	the chosen monitored event triggered a
Deep Samplesin the call stack when the chosen monitored event triggered a sample collection.% of Deep SamplesPercentage of deep samples out of the total count of the samples gathered100*(Deep Sample)/(Total Sample collected)No. of PathsNumber of unique paths containing this function.Source FileThe source files containing the function. The line number is shown in parentheses.ModuleThe name of the module that contains this function. Full path is available on tooltips		sample collection.
SamplesIn the call stack when the chosen monitored event triggered a sample collection.% of Deep SamplesPercentage of deep samples out of the total count of the samples gathered100*(Deep Sample)/(Total Sample collected)No. of PathsNumber of unique paths containing this function.Source FileThe source files containing the function. The line number is shown in parentheses.ModuleThe name of the module that contains this function. Full path is available on tooltips	Doop	Number of times this function was present
Image: Provent triggered a sample collection.% of Deep SamplesPercentage of deep samples out of the total count of the samples gathered100*(Deep Sample)/(Total Sample collected)No. of PathsNumber of unique paths containing this function.Source FileThe source files containing the function. The line number is shown in parentheses.ModuleThe name of the module that contains this function. Full path is available on tooltips	-	in the call stack when the chosen monitored
% of Deep Samplescount of the samples gathered100*(Deep Sample)/(Total Sample collected)No. of PathsNumber of unique paths containing this function.Source FileThe source files containing the function. The line number is shown in parentheses.ModuleThe name of the module that contains this function. Full path is available on tooltips	Samples	event triggered a sample collection.
Samples100*(Deep Sample)/(Total Sample collected)No. of PathsNumber of unique paths containing this function.Source FileThe source files containing the function. The line number is shown in parentheses.ModuleThe name of the module that contains this function. Full path is available on tooltips		Percentage of deep samples out of the total
Image: No. of PathsNumber of unique paths containing this function.Source FileThe source files containing the function. The line number is shown in parentheses.ModuleThe name of the module that contains this function. Full path is available on tooltips	% of Deep	count of the samples gathered
No. of PathsNumber of unique paths containing this function.Source FileThe source files containing the function. The line number is shown in parentheses.ModuleThe name of the module that contains this function. Full path is available on tooltips	Samples	100*(Deep Sample)/(Total Sample
No. of Pathsfunction.function.The source files containing the function. The line number is shown in parentheses.ModuleThe name of the module that contains this function. Full path is available on tooltips		collected)
function.Source FileThe source files containing the function. The line number is shown in parentheses.ModuleThe name of the module that contains this function. Full path is available on tooltips	No. of Poths	Number of unique paths containing this
Source FileThe line number is shown in parentheses.The name of the module that contains thisModulefunction. Full path is available on tooltips		function.
The line number is shown in parentheses.The name of the module that contains thisModulefunction. Full path is available on tooltips	Sourco Eilo	The source files containing the function.
Module function. Full path is available on tooltips	Source Pile	The line number is shown in parentheses.
The second secon		The name of the module that contains this
text of the corresponding cell of the table.	Module	function. Full path is available on tooltips
		text of the corresponding cell of the table.

Parent Table – Appears on the left side of the two tables in the middle of the page. It shows the parents (the functions that call the current function) of the selected function in the function table. This table is updated when changing the selection on function table.

NO/ 1
0% classic.exe

This table has the following columns:

	Lists all the function names that called the
	current function. In the call stack samples,
Parents	those parents (direct ancestors) are
	immediately below the currently selected
	function.
Complee	The distribution of the currently selected
Samples	function's deep-samples between its parents.
% of	Percentage of the paths samples from the total
% 01 Samples	deep samples of the currently selected
Samples	function.

Children Table – Appears in the right side of two tables in the middle of the page. It shows the Children (the functions called by the current function) of the selected function in the function table. This table is updated on changing the selection on function table.

Self + children	Samples	% of samples	Module	
multiply_matrices(void)	10150	99.90%	classic.exe	
initialize_matrices(void)	10	0.10%	classic.exe	
[self]	0		classic.exe	

This table has the following columns:

Lists all of the function names called by the

Self + children	current function. In the call stack samples, those children (direct descendants) are immediately above the current function. It also lists one entry called [self] . This entry is on the selected function in the function table.
Samples	The distribution of the currently selected function's deep-samples between its children. For the [self] function it is the self-samples (as there are no descendants to the function when it has been sampled itself).
% of Samples	Percentage of the paths samples from the total deep samples of the currently selected function.

Path Graph Table – Represents the call chain in the form of a tree. In the tree, a child node represents a function called by the function represented by parent node.

Paths			Show Call Graph s	election path
Function	Self samples	Downstream samples	% of Downstream samples	Module
4 RtlInitializeExceptionChain		10160	100.00%	ntdll.dll
# BaseThreadInitThunk		10160	100.00%	kernel32.dll
mainCRTStartup		10160	100.00%	classic.exe
▷ main		10160	100.00%	classic.exe

This table has the following columns:

•

	6					
Function	The names of functions represented in					
1. alle fioli	the call chain tree.					
Self samples	Number of samples in this function.					
Downstream-	From this point down in the path, not					
samples	including the function samples.					
% of	Percentage of downstream-samples out					
Downstream-	of all samples for the specific call path.					
samples	or an samples for the specific call path.					

 Show Call Graph selection path – This checkbox controls whether to show a line that connects all the functions in the sub-tree that form a unique path, ending in the selected function. The small number at the end of the path's line is the length of the path (starting with 0).

The path of the selected function is colored in yellow, while the path of the function on which the cursor is currently hovering above is colored in red.

Paths				🚺 Show Call Graph s	election path	
Function		Self samples	Downstream samples	% of Downstream samples	Module	
A BaseThreadInitThun	k		10160	100.00%	kernel32.dll	
mainCRTStartup			10160	100.00%	classic.exe	
nain 🖌			10160	100.00%	classic.exe	
initialize_	matrices(void)	4	6	0.10%	classic.exe	
▲ rand			6	0.06%	classic.exe	
	getptd	6		0.06%	classic.exe	
http://	matrices(void)	10150		99.90%	classic.exe	

Filters

 Processes – List all the processes in processname(process-id) format

Functions			D	isplay: <u>All Mod</u>	<u>ules</u>	Process: dassic.exe(492) Monitored event: DC accesses	•
Function (21 functions)	Self Samples	Deep Samples	% of Deep Samples	No. of Paths	Source File	Module	
RtlInitializeExceptionChain		4275	99.98%	13		ntdll.dll	
BaseThreadInitThunk		4275	99.98%	13		kernel32.dll	-
multiply_matrices(void)	4081	4081	95.44%	1	classic.cpp(59)	classic.exe	1
initialize_matrices(void)	58	58	1.36%	1	classic.cpp(36)	classic.exe	
_getptd	36	58	1.36%	4	tidtable.c(289)	classic.exe	

It also has the **All Processes** option. If a process is selected in this combo-box, all the functions from that process are listed in the table. If **All Processes** is selected,

then all the functions from all the modules of the current session are displayed in the current table.

Functions			D	isplay: <u>All Mod</u>	ules	Process: classic.exe(492) Monitored event: DC accesses	v
Function (21 functions)	Self Samples	Deep Samples	% of Deep Samples	No. of Paths	Source File	Module	
RtlInitializeExceptionChain		4275	99.98%	13		ntdli.dli	
BaseThreadInitThunk		4275	99.98%	13		kernel32.dll	
multiply_matrices(void)	4081	4081	95.44%	1	classic.cpp(59)	classic.exe	1
initialize_matrices(void)	58	58	1.36%	1	classic.cpp(36)	classic.exe	
_getptd	36	58	1.36%	4	tidtable.c(289)	classic.exe	

Monitored event – Combo-box at the top right

This lists the set of monitored events observed during current profiling. Only the samples, that were collected when the event selected in this combo-box was triggered, are listed.

Display Settings dialog – Opened from the hyperlinklabeled Display above the table

Functions			D	isplay: <u>All Mod</u>	ules	Process: dassic.exe(492) Monitored event: DC accesses
Function (21 functions)	Self Samples	Deep Samples	% of Deep Samples	No. of Paths	Source File	Module
RtlInitializeExceptionChain		4275	99.98%	13		ntdll.dll
BaseThreadInitThunk		4275	99.98%	13		kernel32.dll
multiply_matrices(void)	4081	4081	95.44%	1	classic.cpp(59)	classic.exe
initialize_matrices(void)	58	58	1.36%	1	classic.cpp(36)	classic.exe
_getptd	36	58	1.36%	4	tidtable.c(289)	classic.exe

Only one checkbox, **Display system modules**, is enabled when opened from **Call Graph view**, which is used to show/hide the system modules in the view.

Context Menu

The context menu provides the same information for each table discussed above, except Display Call Graph details, which are not available in the Function Table.

Paths	Paths 🔲 Show Call Graph selection path								
Function Self sample	s Downstrear	m samples	% of Downstream sample	s Module					
A RtIInitializeExceptionChain	4081		100.00%	ntdil.dli					
RtIInitializeExceptionChain	4081		100.00%	ntdil.dli					
BaseThreadInitThunk	4081		100.00%	kernel32.dll					
multiply_matrices(void) 4081			Source Code	classic.exe					
		Displa	ay in Functions View						
		Сору							
		Select							
		Expan	nd All						
		Colla	pse All						

Context menu items consist of the following.

Open Source Code	Opens the source code or disassembly view for the currently selected function in the current table.
Display in Call Graph	Opens the Call Graph view for the currently selected function in the current table. This menu item is not available for Function table .
Display in Functions View	Opens the Function view for the currently selected function in the current table. This menu item is not enabled for functions that are not present in Functions View (this can happen if the function was back-traced from another function (that may have been actually sampled), but not sampled – the monitored event was never triggered directly for that function).
Сору	Copies the selected column(s) in the current table, along with the headers to the clipboard. See the common context menu section.

Expand All	Expands all the nodes below the selected function's node in the tree.
Collapse All	Collapses all the nodes below the selected function's node in the tree.

© 2015 Advanced Micro Devices. All rights reserved.



Profile Session Source or Disassembly View

The Source Code / Disassembly view shows the source lines annotated with assembly instructions and sample count for a selected function.

Opening the Source Code View

✓ ▲ Apr-10-2016 [13-00-28 Image: Overview > 105 Image: Overview > 106 Image: Overview > 108 Image: Overview	deXL Explorer 🛛 🗗 🗙	Ö Apr-10)-2016_13-0	0-20 (CPU: Time-based Sampling) 🗵 🛛 👌 Apr-10-2016_13-00-23 (CPU: Time-based Sampling) 🔝	A Apr-1	0-2016_13-00-28 (CP	U: Assess Performance)	3
V © CPU: Time-based Sampling Control (12/10/2) classic (12/10/2		0 Profile	Overview	🖞 Modules 🗵 📱 Call Graph 🗵 🕼Release \AMDTClassidMatMul.exe - Source/Disass	embly 🗵			
		c:\jenkins\	workspace\c	odex/-2.0/release/codex/12.0/codex/1examples/amdtclassicmatmul/amdtclassicmatmul.cop				
Image: Construction is sourced Construction is sourced in the source code Code Bytes Hotspot Samples % of Hotspot Samples							Hotspot Indicator Time	4
Wodules 0.028 yrsts you nutropy samples source code 0.028 yrsts you nutropy samples source code Y Source Code 95 /// Multiply the two matrices Y Source Code 96 for (int : 0; i < COULMNS; i++)	773							
I Call Gaph > 94 0x121000 { // Multiply metrices() I Functions > 95 0x121008 for (int : 0; i < 0x1VS; i++)		Line	Address		Code Bytes	Hotspot Samples	% of Hotspot Samples	Tim
I Functions 7 94 00/2010 { 1 Source Code 5 /// Multiply the two matrices Y @ Apr-10-2016 [3-00-23 9 9 { { Ø Modules 9 9 { { I Functions 9 9 { { Ø Modules 9 9 { { I Functions 10 0 0121116 float sum = 0.0; I Functions 10 0 0121116 float sum = 0.0; I Functions 10 0 0121116 float sum = 0.0; I Functions 10 0 0121116 float sum = 0.0; I Functions 10 0 0121112 sum = sum + matrix_d[0][1] * matrix_d[0][1]; V A Apr-10-2016 [3-00-28 10 0 0121112 sum = sum + matrix_d[0][1] * matrix_d[0][1]; Ø Overview > 10 0 0121120 sum = sum + matrix_d[0][1] * matrix_d[0][1] * matrix_d[0][1]; Ø Overview > 10 0 0121120 sum = sum + matrix_d[0][1] * matri								
y iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii		> 94	0x12f10f0					
Image: AddTCLassidMatMulace > 98 0x12f100 for (mt = 0; i < ROUNS; i++)								
v ∂. Apr-10-2016 [13-00-23 > 98 0.1211100 for (int j = 0; j < COLUMINS; j++)			0x12f10f8	for (int i = 0 ; i < ROWS ; i++)				
0 Overview > 50 01/21/100 For (m(t) = 0; t < COLUMNS; t++)								
99 (2 100 0.00% 3 1 Functions 101 for (int k = 0; k < COLUMNS; k++)			0x12f1100	for (int j = 0; j < COLUMNS; j++)				
 I call Graph I functions I functions I functions I curctions I curc			0.4004446	{			0.000/	2
Image: Second secon	🚊 Call Graph		UXIZTIIIO			3	0.60%	3
 ▲ CPU: Assess Performance > 103 0x12f1122 sum = sum + matrix_a[i][k]* matrix_b[k][j]; 475 95.77% 4 475 95.77% 4 90 Overview > 105 0x12f112 sum; 106 } 106 } 107 } 108 0x12f120a } 109 100 109 100 100 void improved_multiply_matrices() 111 { 112 // Multiply the two matrices 113 for (int i = 0; i < ROWS; i++) 114 { 115 for (int k = 0; k < COLUMNS; k++) 116 { 117 for (int j = 0; j < COLUMNS; j++) 	1177			for (int k = 0; k < COLOMINS; k++)				
✓ ▲ Apr-10-2016_13-00-28 104 } ◎ Overview > 105 0x12f11d9 matrix_r[i][j] = sum ; 18]3.63% 1 ◎ Modules 106 }	✓ ▲ CPU: Assess Performance		0v12f1122	(cum = cum + matrix slift) * matrix kft/fil)		175	05 779/	475
O Overview > 105 0x12f11d9 matrix_r[i][j] = sum; 18 3.63% 1 Image: Call Graph 107 } 107 } 107 } 108 0x12f120a } 109 109 100 100 111 { 112 // Multiply the two matrices() 111 { 112 // Multiply the two matrices 113 for (int i = 0; i < ROWS; i++)	✓ ▲ Apr-10-2016_13-00-28		UXIZITIZZ	sum = sum + maux_a(r)(k) - maux_b(k)() ;		4/J	90.77%	4/3
Modules 106 } Image: Call Graph 107 } Image: Functions > 108 0x12f12Da 109 109 109 110 void improved_multiply_matrices() 111 { 112 // Multiply the two matrices 113 for (int i = 0; i < ROWS; i++)	Overview		0v12f11d0	natrix rfilli) – cum :		18	2 62%	18
Image: Call Graph 107 } Image: Functions > 108 0x12f120a 109 100 void improved_multiply_matrices() 111 { 112 // Multiply the two matrices 113 for (int i = 0; i < ROWS; i++)	🖗 Modules		VAT211103) 1160.1x_[(]()] - 360.1,		10	2.03 %	10
Image: Second	📕 Call Graph) }				
109 110 void improved_multiply_matrices[) 111 { 112 // Multiply the two matrices 113 for (int i = 0; i < ROWS; i++)	§ Functions		0x12f120a	}				
110 void improved_multiply_matrices() 111 { 112 // Multiply the two matrices 113 for (int i = 0; i < ROWS; i++)			0.1211200	,				
111 { 112 // Multiply the two matrices 113 for (int i = 0; i < ROWS; i++)				void improved multiply matrices()				
113 for (int i = 0; i < ROWS; i++)				{				
113 for (int i = 0; i < ROWS; i++)		112		// Multiply the two matrices				
114 { 115 for (int k = 0; k < COLUMNS; k++) 116 { 117 for (int j = 0; j < COLUMNS; j++)		113						
116 { 117 for (int j = 0; j < COLUMNS; j++)		114						
117 for (int j = 0; j < COLUMINS; j++)		115		for (int $k = 0$; $k < COLUMNS$; $k++$)				
		116		{				
		117		for (int j = 0 ; j < COLUMNS ; j++)				
(118)		118		1				>

This view can be opened from one of the following ways:

• **Overview Page**: Double-click on a function in the

functions table, or right-click on a function, and select **Open Source Code**.

- **Functions View**: Double-click on a function in the functions table, or right-click on a function, and select **Open Source Code.**
- **Call-Graph View**: Right-click on a function in the functions, and select **Open Source Code**.

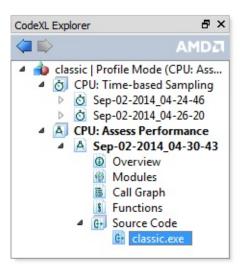
5 Hottest Functions									
Function		Samples	% of Hotspot Samples	s Modul	le				
multiply_matrices(void)		975	64.10%	classic	exe				
C initialize_matrices(void) getptd memcpy QListData:dispose(struct QLis other	stData::Data *)	28 25	3.16% 3.02% 1.84% 1.64% 26.23%	classic classic win32k Qt5Co	xexe D k.sys D re.dll C	pen Source Code isplay in Call Graph isplay in Functions \ opy elect All			
Profile Overview 🔒 Call Gray	ph 🗵								
	ph 🛛		D	splay: <u>Al Mod</u>	ules	Process: da	ssic.exe(492)	Monitored event	DC accesses
Functions		Deep Samples	Di % of Deep Samples			Process: da Module	ussic.exe(492)	Monitored event	DC accesses
Functions	Self Samples	Deep Samples 4275					ssic.exe(492)	Monitored event	DC accesses
Functions Function (21 functions) S	Self Samples		% of Deep Samples	No. of Paths		Module			DC accesses
Functions iunction (21 functions) S RtIInitializeExceptionChain BaseThreadInitThunk	Self Samples	4275	% of Deep Samples 99.98%	No. of Paths 13		Module ntdll.dll kernel32.dll	Open Se	ource Code	DC accesses
Functions iunction (21 functions) S RtIInitializeExceptionChain BaseThreadInitThunk	Self Samples	4275 4275	% of Deep Samples 99.98% 99.98%	No. of Paths 13	Source File	Module ntdll.dll kernel32.dll classic.exe	Open Se		DC accesses
Aunctions Function (21 functions) S RtlInitializeExceptionChain BaseThreadInitThunk multiply_matrices(void) 4 initialize_matrices(void) 5	Self Samples	4275 4275 4081	% of Deep Samples 99.98% 99.98% 95.44%	No. of Paths 13	Source File classic.cpp(59)	Module ntdll.dll kernel32.dll classic.exe	Open Se	ource Code	DC accesses
Functions Function (21 functions) S RtIInitializeExceptionChain BaseThreadInitThunk multiply_matrices(void) 4 initialize_matrices(void) 5 _getptd 3	Self Samples	4275 4275 4081 58	% of Deep Samples 99.98% 99.98% 95.44% 1.36%	No. of Paths 13	Source File classic.cpp(59) classic.cpp(36)	Module ntdlLdll kernel32.dll classic.exe classic.exe	Open So Display i	ource Code n Functions View	DC accesses
Functions Function (21 functions) S RtflnitializeExceptionChain BaseThreadInitThunk multiply_matrices(void) 4 initialize_matrices(void) 5 _getptd 3	Self Samples 1081 18 16 16	4275 4275 4081 58 58	% of Deep Samples 99.98% 99.98% 95.44% 1.36% 1.36%	No. of Paths 13	Source File classic.cpp(59) classic.cpp(36) tidtable.c(289)	Module ntdlLdll kernel32.dll classic.exe classic.exe classic.exe	Open So Display i Copy	ource Code n Functions View	DC accesses

Open Source Code View from Overview Page

Open Source Code View from Call Graph View

After selecting a function and clicking **Open Source Code**, a source code view will be created.

A separate instance of source code view is opened for each module. When the source code is created, a node is added in the CodeXL Explorer, called **Source Code.** Under this node, a node will be created for the module's source code view.



If CodeXL cannot locate the source file, a file selection dialog is opened, and the user will be asked to locate the source file. If the source file will not be located, the source code view will display only disassembly.

On the top of the source code view, a combo-box is created with the list of the module's functions. The function that was rightclicked will be selected in the functions combo-box, and will be highlighted in the code.

		sic'¦dassic.cpp) - 0x401195] : multiply_matrices(void) Display: <u>All Data, All Modules</u>			Hotspot Indicator	ata cache access	es	
Line	Address	Source Code	Code Bytes	Hotspot Samples	% of Hotspot Samples	DC accesses	DC misses N	lisaligi
58		void multiply_matrices()						
Þ 59	0x4010c0	{						
60		// Multiply the two matrices						
⊳ <mark>6</mark> 1	0x4010c4	for (int i = 0 ; i < ROWS ; i++) {						
▷ 62	0x4010d0	for (int j = 0 ; j < COLUMNS ; j++) {						
⊳ <mark>6</mark> 3	0x4010e1	float sum = 0.0 ;		8	0.20%	8	28	
64		for (int k = 0 ; k < COLUMNS ; k++) {						
⊳ <mark>65</mark>	0x4010f0	<pre>sum = sum + matrix_a[i][k] * matrix_b[k][j];</pre>		4,068	99.68%	4,068	5,124	
66		}						
Þ 67	0x40116a	matrix_r[i][j] = sum ;		5	0.12%	5	18	
68		}						
69		}						
Þ 70	0x401194	}						
71								
72		void print_elapsed_time()						
73								
74		double elapsed ;						
75		double resolution ;						
76								l
77		// Obtain and display elapsed execution time						
78		elapsed = (double) clock() / CLK_TCK ;						
79		resolution = 1.0 / CLK_TCK ;						
80								
81		fprintf(result_file,						

Source Code View opened for "multiply_matrices" function

Data Displayed in Source Code View

The source code consists of a table displaying the following data for each source code line / disassembly line in the current displayed source file:

Line	The source code line number.
Address	Memory address where each instruction is located.

Source Code	The source code for the current line.
Code Bytes	Byte representations of the actual machine instructions.
Hotspot Samples	The amount of samples collected for the current hotspot indicator event / metric. The hotspot indicator is selected in the hotspot indicator combo-box.
% of Hotspot Samples	The precentage of samples collected for the current hotspot indicator event / metric. The percentage is calculated relatively to the currently display function (all the function source code lines will sum up to 100%). The hotspot indicator is selected in the hotspot indicator combo-box.
Samples count for each of the collected / calculated data of the current session	The following columns will contain the amount of samples collected for the monitored events and calculated metrics. Use the Display Settings Dialog to select which of the metrics will be displayed currently in the table.

Functions combo-box



The functions combo-box contain the list of the current module's functions. Selecting a function will highlight the function in the displayed source code, or will display disassembly code for the requested function.

Function	[0x4010c0	- 0x401195] : multiply_matrices(void) Display: <u>All Data, All Modules</u>	Process:		TID:	All Threads Hotspot	Indicator Da	ata cache accesses	•
Line	Address	Source Code	Code	All Processes classic.exe(388)	nples	% of Hotspot Samples	DC accesses	DC misses Misalign acces	is CPU 1
58		void multiply_matrices()		classic.exe(2284) classic.exe(2612)					
⊳ 59	0x4010c0	{		classic.exe(2684)					
60		// Multiply the two matrices							
⊳ 61	0x4010c4	for (int i = 0 ; i < ROWS ; i++) {							
⊳ <mark>62</mark>	0x4010d0	for (int j = 0 ; j < COLUMNS ; j++) {							
⊳ 63	0x4010e1	float sum = 0.0 ;		29		0.21%	29	84	413
64		for (int k = 0 ; k < COLUMNS ; k++) {							
⊳ <mark>65</mark>	0x4010f0	sum = sum + matrix_a[i][k] * matrix_b[k][j] ;		13,660		99.50%	13,660	17,008	223,1:
66		}							
⊳ <mark>67</mark>	0x40116a	matrix_r(i)[j] = sum ;		39		0.28%	39	72	518
68		}							
69		}							
▷ 70	0x401194	}							

Process IDs combo-box

Function	n: [0x4010c	0 - 0x401195] : multiply_matrices(void) Display: <u>All Data, All Modules</u>	Process: All Proc	resses 🔻 TID:	All Threads All Threads	lotspot 1	Indicator Da	ta cache accesses	
Line	Address	Source Code	Code Bytes	Hotspot Samples	2440 Sa	mples	DC accesses	DC misses Misalign access	; CPU
58		void multiply_matrices()			2376 2712				
⊳ 59	0x4010c0	{			2148				
60		// Multiply the two matrices							
⊳ 61	0x4010c4	for (int i = 0 ; i < ROWS ; i++) {							
Þ 62	0x4010d0	for (int j = 0 ; j < COLUMNS ; j++) {							
⊳ <mark>63</mark>	0x4010e1	float sum = 0.0 ;		29	0.21%	1	29	84	413
64		for (int k = 0 ; k < COLUMNS ; k++) {							
⊳ 65	0x4010f0	<pre>sum = sum + matrix_a[i][k] * matrix_b[k][j];</pre>		13,660	99.50%		13,660	17,008	223,1:
66		}							
⊳ 67	0x40116a	matrix_r[i][j] = sum ;		39	0.28%	1	39	72	518
68		}							
69		}							
Þ 70	0x401194	}							

Thread IDs combo-box

Process ID / Thread ID combo-boxes

The process IDs / Thread IDs combo-boxes are displayed only in cases where the displayed module is multi-process or multithreaded. Use the process IDs and thread IDs combo-boxes to display the selected process / thread ID collected samples.

Navigating through the presented data

The navigation can be done using the mouse wheel, the Page Down\Up keyboard keys or the Down\Up Arrow keyboard keys. While you navigate through a large disassembly, CodeXL will fetch another block of disassembly data as you scroll down and reach the bottom. The size of the data block that is being automatically fetched can be configured in the CPU Profiling global settings. To open the CPU Profiling global settings, go to the CodeXL menu bar and click on Tools->Options. Then, choose the CPU Profile tab, and edit the value in the text box to set the data block size in bytes:

Symbols	GPU <u>D</u> ebug	CPU Profile	Analyze				
	itional debug sy	mbol paths					
	mbol Server Dir						
Down	load directory						
	New	<u>R</u> emove			Move Up	Move <u>D</u> own	
	http://msdl.m	icrosoft.com/	download/symbo	S			

Changing the size of the data block which is being automatically fetched when scrolling through disassmbly

Source Code View Display Settings

In order to configure the source code view display settings, click the display settings link on the top of the source code view.

The display settings dialog can be used to select which columns will be displayed in the source code view table, to display the data separately for each core / NUMA node etc'. See more details on the display settings in **Profile Session Display Settings**.

ancorn	[UX40100	0 - 0x401195] : multiply_matrices(void) V Display: All Dat	ta, All Modules	Process: All Proce	sses 🔻 TID: All Thr	eads Hots	pot Indicator Data cache ad	cesses		_
ine	Address	Source Code	Code Bytes	Hotspot Samples	% of Hotspot Samples	DC accesses	DC misses Misalign access	CPU clocks	Ret inst	t
58		void multiply_matrices()								
⊳ 59	0x4010c0	{								
60		// Multiply the two matrices								
⊳ <mark>61</mark>	0x4010c4	for (int i = 0 ; i < ROWS ; i++) {								
⊳ <mark>62</mark>	0x4010d0	for (int j = 0 ; j < COLUMNS ; j++) {								
Þ 63	0x4010e1	float sum = 0.0 ;		29	0.21%	29	84	<mark>41</mark> 3	37	8
64		for (int k = 0 ; k < COLUMNS ; k++) {								
65	0x4010f0	sum = sum + matrix_a[i][k] * matrix_b[k][j] ;		13,660	99.50%	13,660	17,008	223,175	23,456	(
66		}								
67	0x40116a	matrix_r[i][j] = sum ;		39	0.28%	39	72	518	43	5
68		}								
69		}								



Hotspot Indicator combo-box

The hotspot indicator combo-box contains the list of monitored events / metrics that is currently displayed in the source code view. Each of these events / metrics can be selected. The selection of an event / metric will update the '**Hotspot Samples**' and '**% of Hotspot Samples**' columns to contain the data collected for this event / metric. Use the hotspot indicator combo-box to look for the most significant performance bottlenecks of your application.

Function	[0x4010c) - 0x401195] : multiply_matrices(void) 🔻 Display: <u>All Data, All Modules</u>	Process: All Proc	resses TID:	All Threads	Indicator		
Line 58	Address	Source Code	Code Bytes	Hotspot Samples	% of Hotspot Samples	DC access	Misaligned accesses	
50 ▷ 59	0x4010c0	void multiply_matrices() t					CPU clocks not halted (cycles) Retired instructions	
60	0.101000	// Multiply the two matrices					Retired branch instructions Retired mispredicted branch inst	ructions
Þ 61	0x4010c4	for (int i = 0 ; i < ROWS ; i++) {					L1 DTLB and L2 DTLB miss	
62	0x4010d0	for (int j = 0 ; j < COLUMNS ; j++) {						
63	0x4010e1	float sum = 0.0 ;		29	0.21%	29	84	413
64		for (int k = 0 ; k < COLUMNS ; k++) {						
65	0x4010f0	<pre>sum = sum + matrix_a[i][k] * matrix_b[k][j];</pre>		13,660	99.50%	13,660	17,008	223,1
66		}						
> <mark>67</mark>	0x40116a	matrix_r[i][j] = sum ;		39	0.28%	39	72	518
68		}						
69		}						
⊳ 7 0	0x401194	}						

Hotspot Indicator combo-box for Assess Performance Profile Session

Context Menu

The source code context menu can be opened by right-clicking on one of the items in the table.

	Copy Select All
	Expand All Collapse All
 ✓ ✓ 	Show Code Bytes Show Address

The context menu contain the following commands:

Copy the selected row/s in the source code

Сору	view table.
Select All	Select all the rows in the source code view table.
Expand All	Expand all the items in the source code table (show all the disassembly lines)
Collapse All	Collapse all the items in the source code table (hide all the disassembly lines)
Show Code Bytes	Show / hide the code bytes column in the source code view table
Show Address	Show / hide the address column in the source code view table

© 2015 Advanced Micro Devices. All rights reserved.





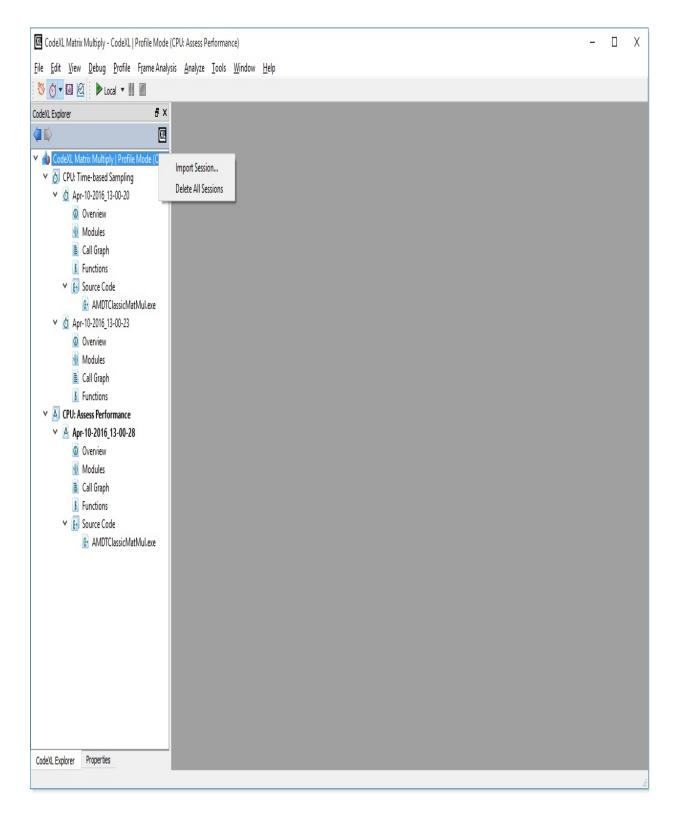
Help > Using CodeXL > CPU Profiler > CPU Profile Data Analysis > Profile Session Display Settings

Profile Session Display Settings

This dialog can be used to configure the display of the CPU profile for the currently opened sessions.

Click on the display link, and open the display settings dialog.

Processes						Display	y: Branch assessment, System Modules Hidden Display by: Processes
Process	PID	Ret inst	Ret branch	Ret misp branch	Branch rate	Mispredict rate	Mispredict ratio
classic.exe	492	7,355	2,903	34	1.91160262	0.01942039	0.10425454



Columns

This section lets you select the group of columns displayed by

the current profiled session. Each of the groups in this combobox is a set of data columns for the current displayed session. The list of groups in this check box depends on the current session profile type.

Display System Modules

Display or hide the data collected from system modules. This option is global and affects the content of each of the opened views.

Show Percentage Bars

When this option is unchecked, the profile session data is displayed as values. When checked, the data is displayed as percentages, and percentage bars are displayed within the tables.

CPU Cores

Select the list of cores for which the results are to be displayed.

Separate Data Per Core / NUMA

Display a separate column for each core / NUMA.

After setting the options in this dialog, click OK. CodeXL updates each of the opened sessions with the global options (percentage and system dll display); it also updates the current view with the local settings. The display filter link on the top of the views contains the current display settings. For example: The following Functions view display settings are: "Branch Assessment, System Modules Hidden, Percentages, 2 Cores" This means that the list of columns displayed for the current session are the Branch Assessment data columns. In the modules table, below the displayed columns, are the columns related to branch assessment. The data is displayed in percentages, and only two cores are selected.

Process	PID	Ret inst	Ret branch	Ret misp branch	Branch rate	Mispredict rate	Mispredict ratio
classic.exe	492	100.00%	100.00%	100.00%	1.90705490	0.01941958	0.10637300

Modules Filter Dialog

The below screenshot contain the display settings link for the Functions View. The right section of the string contain the amount hidden and shown modules in the functions view. Click on the right link to open the modules filter dialog.

Inctions Di	splay: <u>All Data, System Mod</u>	ules Hidden, Percentages	15 modules show	wn, 69 modules hidden P	Process All Processe	es
unction	Module	DC accesses	DC misses	Misalign access	CPU clocks	Ret inst
multiply_matrices(void)	classic.exe	79.79%	98.32%		97.16%	82.96%
initialize_matrices(void)	classic.exe	3.93%			0.38%	2.85%
_getptd	classic.exe	3.76%			0.22%	2.46%
QListData::dispose(struct QListData::Data *)	Qt5Core.dll	2.05%	0.08%		0.22%	3.34%
rand	classic.exe	1.72%			0.12%	0.74%
QBlittablePlatformPixmap::markRasterOverlay(class QPoint	Qt5Gui.dll	0.98%			0.10%	1.18%
qNormalizeRadii(class QRect const &, class QSize const *, cl	Qt5Gui.dll	0.65%			0.09%	1.08%
QPainterReplayer::setupTransform(class QPainter *)	Qt5Gui.dll	0.57%			0.22%	0.15%
QApplication::topLevelWidgets(void)	Qt5Widgets.dll	0.57%	0.16%		0.12%	0.20%
QGraphicsAnchorLayout::'default constructor closure'(void)	Qt5Widgets.dll	0.57%	0.16%		0.06%	0.15%
QRasterPaintEngine::penChanged(void)	Qt5Gui.dll	0.49%			0.03%	0.49%
QHashData::nextNode(struct QHashData::Node *)	Qt5Core.dll	0.41%	0.32%		0.15%	0.74%
QArrayData::data(void)	Qt5Core.dll	0.33%			0.01%	0.05%
QVectorPath::convertToPainterPath(void)	Qt5Gui.dll	0.25%			0.04%	0.54%
QLoggingCategory::categoryName(void)	Qt5Core.dll	0.16%			0.01%	0.10%
QMetaObject::indexOfProperty(char const *)	Qt5Core.dll	0.16%	0.08%		0.03%	0.29%
QFontEngine::harfbuzzFont(void)	Qt5Gui.dll	0.16%			0.01%	0.05%
QTextEngine::shapeTextWithHarfbuzz(struct QScriptItem co	Qt5Gui.dll	0.16%			0.01%	
QRegion::isEmpty(void)	Qt5Gui.dll	0.16%			0.02%	0.20%
CodeXL.exe!0x0016934b	CodeXL.exe	0.08%				
QChar::QChar(char)	Qt5Core.dll	0.08%				
QChar::isHighSurrogate(void)	Qt5Core.dll	0.08%				
QChar::isHighSurrogate(unsigned int)	Qt5Core.dll	0.08%				0.10%
QMutex::unlock(void)	Qt5Core.dll	0.08%				
QListData::append(int)	Qt5Core.dll	0.08%			0.02%	0.05%
QListData::append(void)	Qt5Core.dll	0.08%			0.01%	
QRect::operator&(class QRect const &)	Qt5Core.dll	0.08%				
QUnicodeTables::properties(unsigned int)	Qt5Core.dll	0.08%			0.01%	

Display System modules in Modules Filter

When this box is checked, the table above it includes the system modules. When un-checked, these modules are not shown. This option is enabled when system modules are hidden in all views.

1

The modules table icons represent the modules type.

a a 4	32/64 bit user module
an at	32/64 bit system module

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Data Analysis > Importing Profile Data

Importing Profile Data

The profile data in a CodeXL EBP file can be imported into a CodeXL project. A new session is created for the profile data.

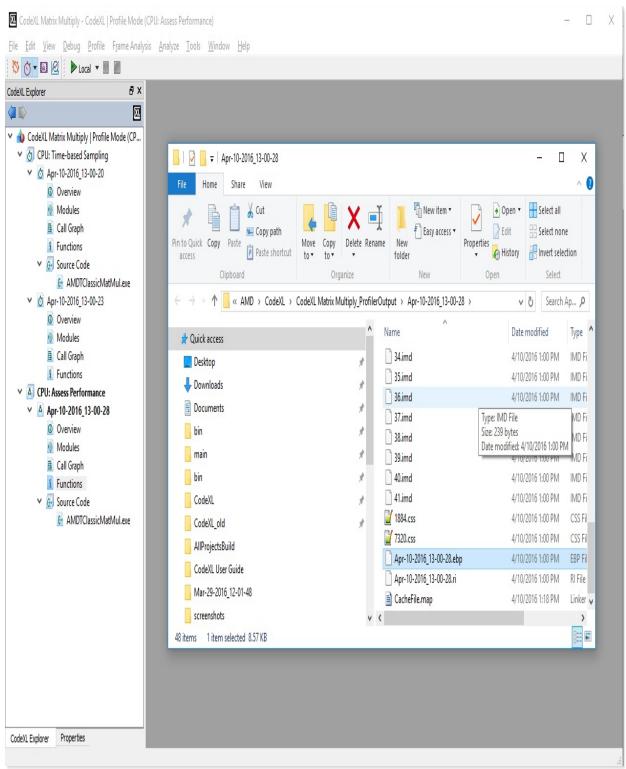
To import profile data:

- 1. If a CodeXL project is not already opened, create a new CodeXL project or open an existing CodeXL project.
- 2. Right-click the project name in the CodeXL Explorer pane. The shortcut menu is displayed.

CodeXL Matrix Multiply - CodeXL Profile Mode (CPU: Assess Performance)	1.00	Х
<u>F</u> ile <u>E</u> dit <u>V</u> iew <u>D</u> ebug <u>P</u> rofile Frame Analysis <u>A</u> nalyze <u>T</u> ools <u>W</u> indow <u>H</u> elp		
🏷 🚫 🕶 🔟 💌 📶		
CodeXL Explorer Ø ×		
V at CodeVI Detro Multiply Deefle Mode (C		
✓ O CPU: Time-based Sampling		
V @ Apr-10-2016_13-00-20 Delete All Sessions		
0 Overview		
Modules		
I Functions ✓ ⊕ Source Code		
Ger AMDTClassicMatMul.exe		
✓ Ambretasachwamatee		
Modules		
E Call Graph		
1 Functions		
V 🛕 CPU: Assess Performance		
✓ Apr-10-2016_13-00-28		
0 Overview		
Modules		
E Call Graph		
v ⊕ Source Code		
Government of the second		
CodeXL Explorer Properties		

Click Import Session.....
 Select the CodeXL EBP file to be imported into the

project.



5. A new session is created and displayed for the imported

profile data.

CodeXL Matrix Multiply - CodeXL | Profile Mode (CPU: Assess Performance) Х File Edit View Debug Profile Frame Analysis Analyze Tools Window Help 🖔 🔿 🔻 🔟 🖉 🕴 🕨 Local 💌 📗 CodeXL Explorer đΧ 🔺 Apr-10-2016_13-00-28 (CPU; Assess Performance) 🔟 🛛 🖄 Mar-29-2016_12-01-48 (Imported - ScimarkStable) (CPU; Time-based Sampling) 🚺 XI Profile Overview 👻 👜 CodeXL Matrix Multiply | Profile Mode (CPU: Assess Performa... **5 Hottest Functions** Display: All Modules Hotspot Indicator Timer ✓ ♂ CPU: Time-based Sampling) 👌 Apr-10-2016_13-00-20 Function Samples % of Hotspot Samples Module ٨ > d Apr-10-2016_13-00-23 1.66% FFT_transform_internal 9,102 21.66% ScimarkStable.exe Mar-29-2016_12-01-48 (Imported - ScimarkStable) ⁶⁴ LU factor ScimarkStable.exe 8,742 20.80% Overview Random_nextDouble 7,323 17.42% ScimarkStable.exe Modules SOR execute ScimarkStable.exe 6,480 15,42% 📱 Call Graph SparseCompRow matmult 6.090 14.49% ScimarkStable.exe § Functions 5 Hottest Modules ✓ ▲ CPU: Assess Performance Module Samples % of Hotspot Samples > A Apr-10-2016_13-00-28 1,969 99.86% ScimarkStable.exe ucrtbase.dll 42 0.10% ⁶⁴ ntoskrnl.exe 13 0.03% 👫 ntdll.dll 2 0.00% Mtfs.svs 0.00% Execution Target Path: C: \CodeXL-Testing \CPUProfilingApps \ScimarkStable \x64\Release \ScimarkStable.exe Working Directory: C: \CodeXL-Testing \CPUProfilingApps \ScimarkStable \x64 \Release Profile Session Directory: C: Users salgrana AppData Roaming AMD CodeXL CodeXL Matrix Multiply_ProfilerOutput Mar-29-2016_12-01-48 **Command Line Arguments: Environment Variables:** CPU Affinity: Oxf Profile Scope: Single Application Call Stack Sampling: True Call Stack Information: Scope - User mode, Depth - 128, Frequency - per 1 samples, FPO - On **Profile Details** Profile Session Type: Time-based Sampling Monitored Events Profile Start Time: Mar-29-2016 12-01-48 Time 🕕 The 5 Hottest Functions usually indicate the most significant performance bottlenecks of your application. Select a different Hotspot Indicator to display hottest functions CodeXL Explorer Properties

 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Data Analysis > Saving Profile Data

Saving Profile Data

You can save the profile data in the tables of different profile session views for later analysis. This data is save in the CSV files.

To save profile data:

- 1. Right click the table having the profile data to be saved. A context-menu is displayed.
- 2. Click 'Select All'
- 3. Right-click and click 'Copy'

deXL Explorer	₽×	Apr-10-2016_13-00-28 (CPU: Assess Per	rformance) 🔝 🛛 💆 Ma	r-29-2016_12-01-48 (Imported - ScimarkStable) (CPU: Time-based Sampling) 🛽	
	X	Profile Overview		and the second is the of the second	
CodeXL Matrix Multiply Profile Mode (C G CPU: Time-based Sampling Apr-10-2016_13-00-20	PU: Assess Performa	5 Hottest Functions		Display: <u>Al Modules</u> Hotspot Indicator Tim	er
 Apr-10-2016_13-00-23 Mar-29-2016_12-01-48 (Import Overview 	ed - ScimarkStable)	#4 FFT_transform_internal 9,102 #4 LU_factor 8,742	s % of Hotspot Samples 21.66% 20.80%	ScimarkStable.exe ScimarkStable.exe	
 Overview Modules Call Graph Functions CPU: Assess Performance Apr-10-2016_13-00-28 	** Son_ckecute	ay in Call Graph View ay in Functions View	ScimarkStable.exe ScimarkStable.exe ScimarkStable.exe		
	Module Same Module Same Module Same ScimarkStable.exe 41,96 ucrtbase.dll 42 Antoskrnl.exe 13 Module 0.03% Module 2 Module 0.00%				
		Execution			
		Target Path: C:\CodeXL-Testing\CPUProfi			
		Working Directory: C: \CodeXL-Testing\C Profile Session Directory: C: Users\salo		ιοιεγιον γκατεασε ΙD\CodeXL\CodeXL Matrix Multiply_ProfilerOutput\Mar-29-2016_12-01-48	
		Command Line Arguments:			
		Environment Variables:			
		CPU Affinity: 0xf		c ope: Single Application	
	Call Stack Sampling: True	Call Stac	k Information: Scope - User mode, Depth - 128, Frequency - per 1 samples, FPO - On		
	Profile Details				
		Profile Session Type: Time-based Samplin	ng Monitore	ed Events	
			8 Timer		

- 4. Open notepad, and paste the copied profile data.
- 5. Save the file as a CSV file (with .csv extension).

Untitled - Notepad					
File Edit Format View Help					
Process, PID, Ret inst, Ret bra classic.exe, 6924, 12178, 5044, devenv.exe, 6752, 2482, 6662, 3 devenv.exe, 6496, 2194, 5668, 3 devenv.exe, 11684, 1944, 5036, System, 4, 908, 708, 22, 2.108 Explorer.EXE, 2276, 602, 1098,	52, 2.39765263 22, 8.65322495, 02, 6.40619469.	, 0.01945578, 0. 0.73136961, 9.2 0.89532214, 7.6	08158215 1798038 614604	ate, Mispredict r	ratio
devenv. exe, 4896, 270, 678, 82		U Profiling 🕨 classic	¥ .	Search classic	٩
CodexL.exe, 1752, 224, 446, 38 svchost.exe, 436, 190, 466, 30		erreining / eissie		7	
chrome.exe, 10660, 182, 396, 5	Organize 🔻 Ne	w folder			⊞ - ()
Smc.exe, 1548, 164, 374, 32, 1	organize ric	w rolder			
conhost.exe, 3492, 142, 346, 2	J Music	 Name 	¹	Date modified	d Type
cptray.exe, 4544, 134, 240, 56 System Idle, 0, 132, 236, 100,	Pictures				E
svchost.exe, 292, 102, 224, 12	Videos		No items mate	ch your search.	
chrome.exe, 5580, 102, 152, 16	Videos			,	
csrss.exe, 500, 58, 84, 10, 0. WmiPrvSE.exe, 11000, 48, 92, 8 CcmExec.exe, 3028, 38, 106, 8,	📜 Computer				
SynTPEnh.exe, 4780, 36, 72, ,	🚢 Local Disk (C:)				
OUTLOOK.EXE, 5792, 30, 46, 8, SmcGui.exe, 4200, 28, 64, 4, 0	👝 Local Disk (S:)				
BCMWLTRY.EXE, 1192, 24, 50, 14 ProtectionUtilSurrogate.exe, 4 P95tray.exe, 4696, 18, 26, , 0 Prot_srv.exe, 1416, 16, 34, 2, iexplore.exe, 4044, 16, 26, 10	🙀 Network	E	III		,
communicator.exe, 4104, 16, 22 WLTRAY.EXE, 4652, 16, 28, 6, 0					
svchost.exe, 116, 14, 32, 8, 0	File name:	my_data.csv			-
chrome.exe, 11188, 12, 28, 12,	Save as type	Text Documents (*.txt)			•
services.exe, 568, 10, 28, 4, svchost.exe, 828, 8, 30, 2, 0.	Save as type.	reac bocuments (act)			
ccSvcHst.exe, 1916, 8, 20, , 0 chrome.exe, 4932, 8, 20, 4, 0. p4v.exe, 6888, 8, 10, 2, 0.025 lsass.exe, 576, 6, 6, 2, 0.066	Hide Folders	Encoding:	ANSI	Save	Cancel

6. To view the profile data, open the CSV file in a a spreadsheet program, such as Microsoft Excel or OpenOffice.org Calc.

X	🛛 ") • (° •	-			my_data	.csv - Microsof	ft Excel				23
F	ile Home	Insert	Page Lay	out Formu	ilas Dat	a Review	View Team		۵	- 6	23
	Ste Vice Calily B	bri <u>I</u> <u>U</u> + + <u>A</u> + Font				· % ,	Conditional Format Format as Table 🔻 Cell Styles 👻 Styles	ting ▼ 📲 Insert ▼ ■ Delete ▼ Erormat ▼ Cells	~	& Find & * Select	
	A1	• (f:	Process							v
1	А	В	С	D	E	F	G	Н	I	J	E
1	Process	PID	Ret inst	Ret branch	Ret misp	Branch rate	Mispredict rate	Mispredict ratio			Γ
2	classic.exe	6924	12178	5044	52	2.39765263	0.01945578	0.08158215			
3	devenv.exe	6752	2482	6662	322	8.65322495	0.73136961	9.21798038			-
4	devenv.exe	6496	2194	5668	302	6.40619469	0.89532214	7.6614604			
5	devenv.exe	11684	1944	5036	238	8.22721672	0.47905275	2.03038311			
6	System	4	908	708	22	2.10854959	0.05504386	0.70373464			
7	Explorer.EXE	2276	602	1098	72	3.63061166	0.22400974	1.74279916			
8	devenv.exe	4896	270	678	82	4.48354912	0.62786824	2.6894455			
9	CodeXL.exe	1752	224	446	38	1.18774331	0.11679089	1.60826468			
10	svchost.exe	436	190	466	30	0.52502328	0.02903076	0.12379469			
11	chrome.exe	10660	182	396	54	1.84686875	0.25797981	0.8790139			
12	Smc.exe	1548	164	374	32	1.78216124	0.11065934	0.35541886			
13	conhost.exe	3492	142	346	20	1.56762242	0.15875152	0.44774246			
14	cptray.exe	4544	134	240	56	0.99517542	0.38684213	2.2316463			
15	System Idle	0	132	236	100	0.37261903	0.19880953	3.43396235			
	sychost eve my_da	ita 🖉	102	22/1	12	0 99//2012/	0.05/91552	<u>0 19198719</u>		•	1
Rea	ady							⊞□ □ 100% 🤆)

 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Command Line Interface

CPU Profile Command Line Interface

CodeXL CPU Profiler provides a command line interface utility for users who prefer to use command interpreters like **cmd.exe** on Windows and **bash** on Linux. This CLI utility will be used to collect and analyze the profile data. It can also be used from a batch file or a test script.

Usage: CodeXLCpuProfiler.exe command <options> [<InputApplication>] [<InputApplication's command line arguments>]

Following commands are supported:

collect	Run the given input application and collect cpu profile samples.
report	Process the given cpu profile data file and generate a cpu profile report in CSV format.

Following options are supported with **collect** command.

	Predefined profile type to be used to collect samples. Supported profile types are:
	tbp : Time-based Sampling
	assess: Assess Performance
	branch: Investigate Branching
-m <profile type=""></profile>	data_access : Investigate Data Access
	ibs : Instruction-based Sampling

	inst_access: Investigate Instruction Access
	l2_access: Investigate L2 Cache Access
	clu: Cache Line Utilization (Windows only)
-T <n></n>	Sampling interval for custom time-based profiling (tbp). Sampling interval <n> is specified in milliseconds.</n>
-d <n></n>	Profile duration in seconds.
-o <file name=""></file>	Base name of the output file. If this option is specified, default path will be used. The default path will be %Temp%\Codexl- CpuProfile- <timestamp> on Windows and /tmp/Codexl- CpuProfile-<timestamp> on Linux.</timestamp></timestamp>
-p <pid,pid,></pid,pid,>	Profile existing processes (processes to attach to). Process IDs are separated by comma.
-a	System Wide Profile (SWP). If this flag is not set then the command line tool will profile only the launched application or the PIDs attached.
-G	Enable callstack sampling with default Callstack collection sampling interval and Unwind Depth. The default values are:

	Callstack Collection Sampling Interval: 1
	Unwind Depth: 128
	Enable callstack sampling with user defined callstack collection sampling interval, unwind depth, scope and FPO values. Scope and FPO are Windows only options.
	(Windows only)
	Scope should contain one of these options:
	user : Collect callstacks only for code executed in user space.
-g <interval:depth:scope:fpo></interval:depth:scope:fpo>	kernel : Collect callstacks only for code executed in kernel space.
	all : Collect callstacks for code executed in user space and kernel space.
	To collect missing frames due to omission of frame pointers by compiler.
	fpo : Collect missing callstack frames.
	nofpo : Do not collect missing callstack frames. (default)
-C	Core Affinity Mask. Default affinity is all the available cores. In System-wide profiling, samples are collected only from these cores. In Per-

	Process profile, processor affinity is set for the launched application.
-f	Profile the children of the launched application (i.e. processes launched by the profiled application).
-b	Terminate the launched application after profile collection.
-s <n></n>	Start profiling after the specified delay duration <n> in seconds.</n>
-V	Print version string.
-W	Specify the working directory. Default will be the path of the launched application.
-h	Displays this help information.
-C <custom profile=""></custom>	Path to the custom profile XML file.

Following options are supported with **report** command.

-i <file name=""></file>	Input file name. Either the raw profile data file (. prd on Windows and . caperf on Linux) or the processed data file (.ebp) can be specified.
-o <output dir=""></output>	Output directory in which the processed data file (ebp and imd) will be created. The default path will be %Temp%\ <base-name- of-input-file> on Windows and /tmp/<base-< td=""></base-<></base-name-

	name-of-input-file> on Linux.
-V <view xml=""></view>	Specify the View configuration XML file. All the raw data will be reported.
	Specify the report sections to be generated. Supported report sections are: all : Report all the sections.
	overview: Report Overview section.
-R <section,></section,>	process : Report process details.
-IX < Section,>	module : Report module details.
	callgraph: Report callgraph.
	Options process and module are mutually exclusive. Options module and callgraph are mutually exclusive.
-е	Specify the event index for which callgraph will be generated. This event is also used to find the hot functions in the Overview section.
-I	Ignore samples from System Modules.
-P	Show Percentage.

Examples

• Launch the application classic.exe and collect Time-based profile (TBP) samples:

CodeXLCpuProfiler.exe collect -m tbp -o c:\Temp\cpuprof-tbp classic.exe

 Launch the application classic.exe and collect assess performance profile samples for the duration of 10 seconds:

CodeXLCpuProfiler.exe collect -m assess -o c:\Temp\cpuprof-assess -d 10 classic

 Launch the application classic.exe and collect Instruction Based Sampling (IBS) samples in System wide profile

(SWP) mode:

CodeXLCpuProfiler.exe collect -m ibs -a -o c:\Temp\cpuprof-ibs-swp classic

Collect Time-based profile samples in System wide profile mode for the duration of 10 seconds:

CodeXLCpuProfiler.exe collect -m tbp -a -o c:\Temp\cpuprof-TBP-swp -d 10

• Launch the application classic.exe and collect Time-based profile (TBP) samples. Also enable collecting callstack samples whenever the TBP samples are collected:

 $CodeXLCpuProfiler.exe\ collect\ -m\ tbp\ -G\ -o\ c:\ Temp\ cpuprof\ tbp\ classic.exe$

• Print help:

CodeXLCpuProfiler.exe -v

• Print version string:

CodeXLCpuProfiler.exe -h

Once the raw cpu profile data file is generated, CodeXLCpuProfiler report command can be used to generate CSV report from that raw data file:

CodeXLCpuProfiler.exe report -i c:\Temp\cpuprof-tbp.prd -o c:\Temp\cpuprof-tbp-out

Once the raw CPU profile data file is generated using the command line utility, the results can be viewed within CodeXL using the **Import Session** command in the **CodeXL Explorer.**

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Command Line Interface > Profile Configuration File Format

Profile Configuration File Format

This section describes the XML configuration file passed to -C option. This data collection configuration file describes how CodeXL CPU Profiler is to be configured for data collection. Pre-defined configurations are provided with CodeXL. Advanced users can create their own data collection configuration by writing an XML file. A data configuration XML file contains only one configuration.

The <dc_configuration> and </dc_configuration> tags mark the beginning and end of configuration information within a data collection configuration file.

<dc_configuration>

•••

</dc_configuration>

A collection configuration contains <tbp>, >ebp>, <ibs> or <clu> elements. Each element describes a data collection configuration of type indicated by its element name. A collection configuration should contain one or more non duplicate elements.

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profile > CPU Profile Command Line Interface > Profile Configuration File Format > TBP Collection Configuration

TBP Collection Configuration

The <tbp> and </tbp> tags mark the beginning and end of a time-based profile data collection configuration. This element has the following attributes:

name	Configuration name (string)
interval	Sampling interval in milliseconds (float)

Sample TBP configuration:

<dc_configuration>

<tbp name="time based profile"

```
interval="10.0">
```

<tool tip> Find program hotspots

</tool_tip>

<description> Configuration to identify
where an application is spending its time

</description>

</tbp></dc configuration>

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Command Line Interface > Profile Configuration File Format > EBP Collection Configuration

EBP Collection Configuration

The <ebp> and </ebp> tags mark the beginning and end of an event-based profile data collection configuration. This element has the following attributes:

name	Configuration name (string)
------	-----------------------------

The sampling events are specified using <event> element. One are more events can be specified. The tag <event> </event> mark the beginning and end of an event element. It describes how an individual event counter is configured for data collection.

An event has the following attributes:

select	Event select value (integer)
mask	Unit mask value (integer)
OS	Enables OS sampling (Boolean)
user	Enables user level sampling (Boolean)
count	Sampling period (integer)
edge_detect	Enables edge detect when counting even (Boolean). This is optional.
host	Enables host mode event counting (Boolean). This is optional.
guest	Enables guest mode event counting (Boolean). This is optional.

The values must be validated against the events and specific capabilities supported by the measurement platform. The maximum number of events depends upon the number of counters supported by the platform on which the measurements are taken.

```
<tool_tip> HW PMC events based
```

profiling </tool_tip>

<description> Configuration to find potential issues for investigation

</description>

</ebp>

</dc_configuration>

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Command Line Interface > Profile Configuration File Format > IBS Collection Configuration

IBS Collection Configuration

The <ibs> and </ibs> tags mark the beginning and end of an instruction-based sampling (IBS) data collection configuration. This element has the following attributes:

name	Configuration name (string)
fetch_sampling	Enables IBS fetch sampling (Boolean)
fetch_max_count	Maximum periodic fetch count/sampling period (integer)
op_sampling	Enables IBS Op sampling (Boolean)
op_max_count	Maximum periodic op count/sampling period (integer)
op_cycle_count	Count clock cycles (Boolean).

Sample IBS configuration:

<dc_configuration>

<ibs name="instruction based sampling"

fetch sampling="T"

op sampling="T"

fetch_max_count="250000"

op_max_count="250000"

<tool_tip> Collect data using IBS

</tool_tip>

<description> Configuration to attribute samples to instructions precisely </description>

</ibs>

</dc_configuration>

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Command Line Interface > Profile Configuration File Format > CLU Collection Configuration

CLU Collection Configuration

The <clu> and </clu> tags mark the beginning and end of a Cache line utilization (CLU) data collection configuration. This element has the following attributes:

name	Configuration name (string)
clu_sampling	Enables CLU sampling (Boolean)
clu_max_count	Maximum sampling period (integer)

Sample CLU configuration:

<dc_configuration>

<clu name="cache line utilization"

clu_sampling="T"

clu_max_count="250000"

```
<tool_tip> measure of cache line
utilization of L1 data cache </tool tip>
```

<description> Configuration to find
potential issues related to data locality and

data access pattern.

</description>

</clu>

</dc_configuration>

 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Command Line Interface > Profile Configuration File Format > Miscellaneous tags

Miscellaneous tags

The tags <tool_tip> and </tool_tip> mark the beginning and end of a short **tool tip** description of a configuration. The text between the tags is the tool tip description. It is usually only a few key words with no line breaks.

The tags <description> and </description> mark the beginning and end of a short description of a configuration. The text between the tags is the description. It is usually only a few sentences long and may contain line breaks. Line breaks will be replaced by spaces and runs of spaces will be replaced by single space character.

The valid values for a Boolean attribute is one of the strings "T" or "F" which denotes TRUE and FALSE respectively. The default value for Boolean attribute is "F"

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile C/C++ Inline Functions

CPU Profile C/C++ Inline Functions

CodeXL CPU Profiler reports functions that are inlined and attributes samples which belong to the code region of the instances of inlined functions. In the CodeXL session views the inlined functions are displayed with the word "[inlined]" prefixed to the function name. CPU Profiler can identify the inlined functions only if the target application binary includes the information about the instances of inlined functions. If the target application binary does not contain the information about the inlined functions, then CodeXL CPU Profiler would attribute the samples to the non-inlined caller function.

Overview, Functions View, Call Graph View and Source Code view display information about inlined functions.





Help > Using CodeXL > CPU Profiler > CPU Profile PLT Relocations

CPU Profile PLT Relocations

On Linux platforms, CodeXL CPU Profiler attributes and reports the samples that belong to Procedure Linkage Table (PLT) section. PLT information is generated by compiler, which is used by dynamic linker/loader to link the application with its dependent dynamic libraries. Samples that are attributed to PLT instructions are reported against "[PLT] <function-name>" symbol. For example, an application calling a library function rand() will have an corresponding entry for this function in the PLT section. Samples attributed to this PLT entry will be shown as "[PLT] rand" and samples due to actual rand() function will be attributed to "rand()" function entry in the session views.





Help > Using CodeXL > CPU Profiler > CPU Profile on Virtual Machine > VMware Workstation

VMware Workstation

CPU Profiler supports TBP and EBP on Guest OS running on VMware Workstation 11.0 or later. It is always recommended to use latest version of VMware Workstation. Recent AMD Carrizo processor is not yet supported by VMware Workstation 11.1.x.

To run TBP within guest OS, no additional configuration needed in host OS or guest OS.

To run EBP within guest OS, please ensure the following settings are done:

- Enable Virtualization or SVM (AMD-V) in BIOS settings before booting the host OS.
- Enable AMD-V in guest OS VM settings. Edit virtual machine settings > Hardware > Processors> Virtualization engine > Enable "Virtualize AMD-V/RVI"
- Enable vPMC in guest OS VM settings. Edit virtual machine settings > Hardware
 Processors> Virtualization engine > Enable "Virtualize CPU performance counters"

Known Issues on Windows 7 Host OS:

- When CPU Profiler EBP is running on Windows 7 host OS and a Linux guest OS is launched, crash is observed on Windows 7 due to VMware driver.
- If EBP is performed on Windows 7 host OS and EBP is performed on Linux guest OS simultaneously, then crash is observed on Windows 7 due to VMware driver.

These scenarios work fine when host OS is Windows 8, 8.1 and 10.





Help > Using CodeXL > CPU Profiler > CPU Profile on Virtual Machine > Microsoft Hyper-V

Microsoft Hyper-V

CPU Profiler supports TBP on Windows Host OS, Windows/Linux Guest OS running on Hyper-V.

CPU Profiler supports EBP only on Windows 10 Host and Windows 10 Guest OS (running on Windows 10 Host OS). Please enable Virtualization or SVM (AMD-V) in BIOS settings before booting the host OS.





Help > Using CodeXL > CPU Profiler > CPU Profile on Virtual Machine > Xen Project

Xen Project

CPU Profiler supports only TBP on Windows/Linux OS running on Xen hypervisor.





Help > Using CodeXL > CPU Profiler > CPU Profile on Virtual Machine > Linux KVM

Linux KVM

CPU Profiler supports only TBP on Windows/Linux OS running on KVM hypervisor.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > CPU Profiler > CPU Profile Control APIs

CPU Profile Control APIs

CPU Profiler control APIs allow user to limit the profiling scope to a specific portion of the code within the target application. Usually, when the profiling done, it captures the samples for the complete application, i.e. start of execution till end of the application execution. The control APIs can be used to enable the profiler only for a specific part of application, e.g. a CPU intensive loop, a hot function, etc. The target application need to be recompiled after adding the control APIs within the application.

The control APIs:

// To pause CPU profiling, call one of the below two APIs.

int amdtStopProfiling(amdtProfilingControlMode); // Set mode to AMDT_CPU_PROFILING

int amdtStopProfilingEx(void);

// To resume CPU profiling, call one of the below two APIs.

int amdtResumeProfiling(amdtProfilingControlMode); // Set mode to AMDT_CPU_PROFILING

int amdtResumeProfilingEx(void);

CPU Profiler only profiles the code within each Resume, Stop APIs pair. Refer "CPU Profiler Tutorial" on how to use these APIs, compile your target application and profile only the desired part of code.





Help > Using CodeXL > CPU Profiler > CPU Profile IMIX report generation

CPU Profile IMIX report generation

If you are interested in the hot instructions for a target application then in such IMIX report will be useful. IMIX report generates report on hotspot instructions summary.

Sample IMIX report summary:

Disassembly	Samples Percentage	Samples Count
mov [rsp+08h],rcx	5.54	111
retnq	3.54	71
mov rax,[rsp+08h]	3.09	62
sub rsp,28h	2.79	56
mov [rsp+18h],r8d	2.44	49
mov [rsp+10h],edx	2.34	47

Only CPU Profiler CLI interface supports IMIX report generation. Use option '-R imix' in CLI during report generation to generate IMIX information. Refer CodeXLCpuProfiler help (-h) option to get more details.





Help > Using CodeXL > GPU Profiler

GPU Profiler

The GPU Profiler component in CodeXL is a performance analysis tool that gathers data from the API run-time and GPU for OpenCL^m, HSA or DirectCompute applications.

This information can be used by developers to discover bottlenecks in the application and find ways to optimize the application's performance. The GPU Profiler can also be used as a command-line tool.

Features of the GPU Profiler include:

- Measure the execution time of an OpenCL[™], HSA or DirectCompute kernel.
- Query the hardware performance counters on an AMD Radeon[™] graphics card.
- Use the AMDTActivityLogger API to trace and measure the execution of segments in the program.
- Compare multiple runs (sessions) of the same or different programs.
- Store the profile data for each run in a text file.
- Display the IL/HSAIL and ISA (hardware disassembly) code of the kernel for OpenCL[™] kernels and DXASM code for DirectCompute kernels.
- Show a timeline (including data transfer and kernel dispatch) and an API trace for OpenCL[™] or HSA programs.
- Calculate and display kernel occupancy info, which estimates the number of in-flight wavefronts on a compute unit as a percentage of the theoretical maximum number of wavefronts that the compute unit can support.

The following screenshots display the results of the Application Timeline Trace and Performance Counters sessions for an

OpenCL[™] program.

		sis <u>A</u> nalyze <u>T</u> ools <u>W</u> indov	v <u>H</u> elp							- []
CodeXL Explorer	e x	Apr-07-20	16_13-56-5	3 (Application Timeline Trace	.)	X					
	X	Application Timeline Trace	-			_					
 CodeXL Teapot O Application 1 	Profile Mode (Application Fi meline Trace			6503.231 6503.260	.272 6503.289	6503.319	6503.348	6503.377	6503.407	6503.436	6503.
 Apr-07-2 		⊟Host									
d_ APIS		Host Thread 5832									
	ext Summary I Summary	OpenCL		NDRa ueNDRang ueN	ORang Flust	eueWrite eNDR	an iu ueAcquireGL	CopyBuff eueRelea	aseGLCeueMap	E lus lus	WaitForE
	Data Transfer Summary	⊡ OpenCL									
	Kernel Summary	Context 0 (0x0E616350)									
₩ Warn > 0 Apr-07-2	ng(s)/Error(s)	□Queue 1 - Hawaii (0x0E582	998)								
/ Mprorez	10_14-04-03	Data Transfer	,								
		Kernel Execution		eFieldPress eFieldPress eFi	eldPress						Fiel
		Other Enqueue Operation	15								
		Queue 0 - CPU_Device (0xl									
		Data Transfer					(RIT				
		Kernel Execution							nputeIntersection		1
		Other Enqueue Operation				_		con	iputcantel accuon		-
		Outer Enqueue operation	6	12							
				<							
		Host Thread 5832 Summary									
operties	Ð×	CL Top 10 Kernel Summary	•								
omputeIntersection	٨										٨
	Intersection	computeIntersection	0	0	CPU_Device	0.28747	{111}	{NULL}	5832	21833	_
evice CPU_Der		computeIntersection	0	0	CPU_Device	0.26204	{111}	{NULL}	5832	28033	_
	MAND_NDRANGE_KERNEL	computeIntersection	0	0	CPU_Device	0.23858	{111}	{NULL}	5832	35721	
	4 milisecond 7 milisecond	<u>computeIntersection</u>	0	0	CPU_Device	0.23516	{111}	{NULL}	5832	381	
	7 milisecond 3 milisecond	computeIntersection	0	0	CPU_Device	0.23271	{111}	{NULL}	5832	16129	
	D millisecond	<u>computeIntersection</u>	0	0	CPU_Device	0.23076	{111}	{NULL}	5832	32001	v
	nicroseconds										*

. Explorer 🗗	< <u>a</u>	Apr-07	-2016_14-20-10 (GPU; Perform	ance Counters)		×										
	Pe	formance Counters				-										
 CodeXL Teapot Profile Mode (GPU: Perfor Application Timeline Trace Apr-07-2016_13-56-53 API Summary Context Summary Context Summary Kernel Summary Top10 Data Transfer Summary Top10 Kernel Summary Top10 Kernel Summary Warning(s)/Error(s) Apr-07-2016_14-04-03 GPU: Performance Counters 	" 2 3 4 5	Show Zero Columns applySources k1 + applyBuoyancy k2 calculateCurlU k3 applyVorticity k4 + advectFieldVelocity	Hawaii1 Hawaii1 Hawaii1	ExecutionOrder 1 2 3 4 5	ThreadID 7968 7968 7968 7968 7968	CallIndex 271 272 274 276 278	Gli { 6 { 6 { 6 { 6	4 64 4 64 4 64	64} 64} 64}	{ 64 { 64 { 64 { 64	4	1} 1} 1}	Time 0.00933 0.04830 0.04622 0.11170 0.05081	0 0 0	VGPRs 9 7 20 17 46	5 / 21 21 21 21 21 21 21 21
 Apr-07-2016_14-19-49 Apr-07-2016_14-20-10 	6 7		laryCondition k6 Hawaii1 urePrep k7 Hawaii1	6 7	7968 7968	280 282	{ 6 { 6				4		0.06311		8 13	22 3(
	8	<u>computeFieldPress</u>		8 9	7968 7968	284 286	{ 6 { 6			{ 64 { 64		1}	0.02281		8	3{ 3{
	1) computeFieldPress	urelter k8 Hawaii1	10	7968	288	{ 6	4 64	<mark>64</mark> }	{ 64	4	1}	0.02237	0	8	3{
	1	<u>computeFieldPress</u>	urelter k9 Hawaii1	11	7968	290	{ 6	4 64	64}	{ 64	4	1}	0.02104	0	8	3{
	1	computeFieldPress	urelter <u>k8 Hawaii1</u>	12	7968	292	{ 6	4 64	64}	{ 64	4	1}	<mark>0.02118</mark>	0	8	3{
ies 🗗	(1	s <u>computeFieldPress</u>	urelter k9 Hawaii1	13	7968	294	{ 6	4 64	64}	{ 64	4	1}	0.02148	0	8	3{
Profile Session	1	computeFieldPress	urelter <u>k8 Hawaii1</u>	14	7968	296	{ 6	4 64	64}	{ 64	4	1}	0.02267	0	8	3{
• Type Performance Counters	1	computeFieldPress	urelter k9 Hawaii1	15	7968	298	{ 6	4 64	64}	{ 64	4	1}	0.02104	0	8	3{
able Path: C:/Users/Public/Documents/CodeXL/ es/Teapot/Release/AMDTTeaPot.exe	1	computeFieldPress	urelter <u>k8 Hawaii1</u>	16	7968	300	{ 6	4 64	64}	{ 64	4	1}	0.02193	0	8	3{
g Directory: C:/Users/Public/Documents/CodeXL/ es/Teapot/Release	1	computeFieldPress	urelter <u>k9 Hawaii1</u>	17	7968	302	{ 6	4 64	64}	{ 64	4	1}	0.02118	0	8	3{
Scope: Single Application	1	computeFieldPress	urelter k8 Hawaii1	18	7968	304	{ 6	4 64	64}	{ 64	4	1}	0.02237	0	8	3{

 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Profiler > Using the GPU Profiler

Using the GPU Profiler

The GPU Profiler provides two modes:

Application Timeline Trace	This mode provides a high-level overview of 1. An API Trace, showing all OpenCL [™] of H 2. A timeline showing the call sequence and as data transfers (OpenCL [™] only) and kern 3. A set of Summary Pages, providing a set analysis of the application. For more information, see: <u>Application Timeline Trace Session</u> <u>Application Timeline Trace Summary Pi</u>
Performance Counters	This mode collects performance counters fr device by either and OpenCL [™] , HSA or Dir It also displays statistics from the shader cc The performance counters and statistics ca: This mode also can also display the kernel s for an OpenCL [™] , DirectCompute kernel dis For more information, see: <u>GPU Profiler Performance Counters Ses</u> <u>GPU Profiler Code Viewer</u> .

For OpenCL[™] programs, both profiling modes can also generate Kernel Occupancy information for each kernel dispatched to a GPU. For HSA applications Kernel Occupancy information is only available in Performance Counter mode. For more information, see <u>GPU Profiler Kernel Occupancy</u> <u>Viewer</u> and <u>GPU Profiler Kernel Occupancy</u>

To use the GPU Profiler:

1. Create a new project, or open an existing project.

- Switch to Profile Mode in CodeXL. You can switch to Profile Mode using the <u>CodeXL Toolbar</u> or the CodeXL menu.
- 3. In Profile Mode, use the menu to select one of the above two modes.
- 4. Start the profile session using the "Start CodeXL Profiling" toolbar or menu item. Profiling results are gathered while the application is running.

Once the application terminates, a new session is added to the **CodeXL Explorer**. The results of the profile also are displayed by CodeXL.

The following links provide more information on the features available in the GPU Profiler.

Application Timeline Trace Session

GPU Profiler Performance Counters Session

GPU Profiler Summary Pages

<u>GPU Profiler Code Viewer</u>

<u>GPU Profiler Kernel Occupancy Viewer</u>

GPU Profiler Kernel Occupancy

GPU Profile Project Settings

Description of Output Files

Description of Configuration Files

Using the Command-Line Interface

AMDTActivityLogger Library

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Profiler > Using the GPU Profiler > Application Timeline Trace Session

Application Timeline Trace Session

The following screenshot shows the timeline and API trace data for a profile session. To get the .atp file of the result, right-click the session in the <u>CodeXL Explorer</u>, and select "Open Containing Folder" from the menu. See <u>Description of</u> <u>Output Files</u> for a detailed description of the format of this file.

<u>File E</u> dit <u>V</u> iew	-	Performance Counters) alysis <u>A</u> nalyze <u>T</u> ools <u>W</u> indo	v Help	- 0	Х
CodeXL Explorer	Local 🔹 📗 📰	X Apr-07-2016_14-20-10 (GF	J: Performance Counters) 🗵 🧧 Apr-07-2016_13-56-53 (Application Timeline Trace) 🛛		
	[An el ministre counced) 🔤 👘 vip or les se les de le piperceter i mestre rices 📒		
	pot Profile Mode (GPU: Perfor tion Timeline Trace		361.455 0.000 1285.947 2571.893 3857.840 5143.787 6429.734 7715.680	9001.627 1028	87.5
	-07-2016_13-56-53	⊟Host			
	API Summary Context Summary	Host Thread 5832			
	Kernel Summary	OpenCL			
	Top10 Data Transfer Summary	⊡ OpenCL			
	Top10 Kernel Summary	Context 0 (0x0E616350)			
	Warning(s)/Error(s) -07-2016_14-04-03	Queue 1 - Hawaii (0x0E58	998)		
_	rformance Counters	Data Transfer			
Apr	-07-2016_14-19-49	Kernel Execution			
🙆 Apr	-07-2016_14-20-10				
		Other Enqueue Operation			
		Queue 0 - CPU_Device (0)	16582618)		
		Data Transfer			
		Kernel Execution			
		Other Enqueue Operation			
roperties	ð	x	-		
CQUIRE_GL_OBJ	ECTS	A Host Thread 5832 Summar			_
ame	ACQUIRE_GL_OBJECTS	Call Ind Interface	Parameters Result Device Block Kernel Occupa CP	U Tir Device Time	
evice	Hawaii			156 0.0276	
		6 clReleaseEvent	-	015	
ommand Type	ACQUIRE_GL_OBJECTS	6 clReleaseEvent	-	015	
ueued Time	9599.769 millisecond	6 clFlush		044	
lbmit Time	9599.799 millisecond			132 0.0093	
art Time	9600.326 millisecond			127 0.1017	
nd Time	9600.326 millisecond	6 clFlush		044	
				186	
uration	0 nanoseconds	 6 clEnqueueCopvBuff 	r 0x0E582998/0x0EEB3650/0x1CC08D50/0/10.0.01/164.64.641/0/NULL/NULL_CL_SUCCESS4.0_MB_COPY0.0	152 0.0433	

From the application trace data, you can:

 \cdot $\,$ Discover the high-level structure of the application with

the Timeline View. For OpenCL[™] programs, you can use this view to determine the number of OpenCL[™] contexts and command queues created, as well as the relationships between these items in the application.

- Determine if an OpenCL[™] application is bound by kernel execution or data transfer operations.
- View and debug the input parameters and output results for all API calls made by the application with the API Trace View.
- View and analyze the performance for sections in the program, using AMDTActivityLogger.

The panel is divided into two sections. The upper section shows the application timeline, the lower section shows the API trace.

Application Timeline Trace

The application timeline provides a visual representation of the execution of the application. Along the top of the timeline is the time grid, which shows the total elapsed time, in milliseconds, of the application. Timing begins when the first OpenCLTM of HSA call is made by the application; it ends when the final OpenCLTM or HSA call is made.

Directly below the time grid, each host (OS) thread that made at least one OpenCL[™] or HSA call is listed. For each host thread, the API calls are plotted along the time grid, showing the start time and duration of each call. Below the host threads, an API-specific tree shows device-specific information. For OpenCL[™], the tree shows all contexts and queues created by the application, along with data transfer operations and kernel execution operations for each queue. For HSA, the tree shows all kernels dispatched to a particular device.

The Timeline View can be useful for debugging your OpenCL[™] application. Using the data displayed in the timeline, you can:

Easily confirm that the high-level structure of your application is correct. By examining the timeline, you can

verify that the number of queues and contexts created matches your expectations for the application.

- Confirm that synchronization has been performed properly in the application. For example, if kernel A execution is dependent on a buffer operation and outputs from kernel B execution, then kernel A execution appears after the completion of the buffer execution and kernel B execution in the time grid. It can be hard to find this type of synchronization error using traditional debugging techniques.
- Confirm that the application has been using the hardware efficiently. For example, the timeline shows that nondependent kernel executions and data transfer operations occur simultaneously.

Navigating the Application Timeline

The application timeline provides many ways to view and analyze the profile result: through zooming, navigating, and expanding/collapsing.

Zooming

When first opened, the timeline view is fully zoomed out: the entire application timeline is visible in the timeline. It can be useful to zoom in to specific parts of the timeline in order to better understand the profiling data. As you zoom in and out, the time grid at the top changes to display the timestamp of the currently displayed timeline subsection.

1. **Manual zoom** – Use the mouse wheel to manually zoom in and out. Roll the mouse wheel up to zoom in, and down to zoom out. If using a mouse not equipped with a mouse wheel, or if you prefer to use the keyboard, you can use the plus key to zoom in, and the minus key to zoom out. The current zoom pivot point (displayed as a vertical line over the entire timeline) represents the point in the timeline into which the view is zoomed. The zoom pivot point tracks the mouse cursor as it moves over the timeline. The current timestamp represented by the zoom pivot is displayed as a hint in the grid displayed at the top of the timeline.

- 2. **Zoom into specific API call** To zoom into a particular API call, double-click the API call in the API Trace list.
- 3. **Zoom into specified region** To zoom into a specific region of the timeline, hold down the Control key and drag the mouse to highlight a specific region. When you release the mouse button, the timeline is zoomed into the highlighted region. While you are dragging, hints are displayed in the grid at the top of the timeline, showing the start and end timestamps for the selected region, as well as the duration of the selected region.

Navigation

When the timeline is zoomed in, you can navigate to different parts of the timeline. You can use either the horizontal scrollbar (located along the bottom of the timeline), or you can click and drag the mouse to pan the timeline within the viewable area. You also can use the left or right arrow keys on the keyboard to pan the timeline within the viewable area.

Expanding and Collapsing the timeline tree

When the timeline is first displayed, its tree is fully expanded. You can collapse parts of the tree in order to limit the amount of data shown. Use the tree view controls within the timeline to collapse or expand parts of the timeline tree. When a branch of the tree is collapsed, timeline items from the collapsed subbranches are displayed in the parent branch.

Viewing timeline item details

There are several ways to view more information about items

shown in the timeline view.

- 1. **Tooltip hints** Hover the mouse over a block shown in the timeline, and a tooltip hint appears. It gives additional details about that block.
- 2. **Navigating to the API trace** Click an API block in a "Host Thread" row, and that block is selected in the API Trace. There, additional details for that particular API call are shown. Click an item in the "Data Transfer" or "Kernel Execution" row, and the enqueue API that enqueued the data transfer or kernel execution is selected in the API Trace.

API Trace

The API trace is a list of all the OpenCL[™] or HSA API calls made by the application. Each host thread that makes at least one API call is listed in a separate tab. Each tab contains a list of all the API calls made by that particular thread. For each call, the list displays:

- \cdot the index of the call (representing execution order),
- the name of the API function,
- \cdot $\,$ a semi-colon delimited list of parameters passed to the function, and
- the value returned by the function.

When displaying parameters, the Profiler tries to dereference pointers and decode enumeration values; this is in order to give as much information as possible about the data being passed in, or returned from, the function. Double-clicking an item in the API Trace list displays and zooms into that API call in the Host Thread row in the Application Timeline.

For OpenCL[™] Enqueue API calls that result in either a kernel execution or a data transfer operation, there is a clickable entry in the "Device Block" column. Clicking this entry zooms into the corresponding timeline block under the OpenCL[™] tree

in the timeline.

For OpenCL[™] Enqueue API calls that result in a kernel execution on a GPU, there is a clickable value in the "Kernel Occupancy" column. Clicking this entry opens the <u>GPU</u> <u>Profiler Kernel Occupancy Viewer</u>, which provides more information about the <u>kernel occupancy</u>.

If the option to Enable navigation to source code is checked on the <u>Application Timeline Trace page</u>, you can right-click any item in the API trace and choose **Go to source code** from the context menu. This uses the symbol information generated during the trace to navigate to the source code location of the API call. Note that this feature only works if the profiled application was built with debugging information.

The API Trace lets you analyze and debug the input parameters and output results for each API call. For example, you can easily check that all the API calls are returning CL_SUCCESS (OpenCL[™]) or HSA_STATUS_SUCCESS (HSA), or that all the OpenCL[™] buffers are created with the correct flags. You also can identify redundant API calls using the API Trace.

Colors

Colors are used in both the application timeline and the API Trace to help distinguish data transfer and kernel dispatch. Green is used on kernel dispatch items for both the OpenCLTM Enqueue calls from the host and the kernels themselves on the device. Shades of blue are used to color OpenCLTM data transfer items, with slight color variations for read, write, and copy calls.

Note: Special case when the OpenCL[™] trace may be incomplete

When the setting to write trace data in intervals during program execution is enabled on the <u>Application Timeline</u> <u>Trace page</u> (which is the default and only supported mode on

Linux), the application trace might not include the full trace of all APIs called by the application. This is because any APIs called after the final interval in the application's lifetime might be omitted. To limit the number of APIs omitted in this scenario, the Profiler also writes all queued-up trace data when the **clReleaseContext** API is called. However, if an application does not call **clReleaseContext** to clean up any OpenCL[™] contexts it has created, or if it calls any OpenCL[™] APIs after the final **clReleaseContext** call, then the trace might not contain all APIs called.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Profiler > Using the GPU Profiler > GPU Profiler Performance Counters Session

GPU Profiler Performance Counters Session

The following panel shows the GPU performance counters for an OpenCL[™] profile session. To get the .csv file of the result, right-click the session in the <u>CodeXL Explorer</u>, and select "Open Containing Folder" from the menu. See <u>Description of</u> <u>Output Files</u> for a detailed description of the format of this file.

_	how Zero Column																	
	Method	ExecutionOrder	ThreadID	CallIndex	Glo	balWor	kSize	Wor	Gr	oupSize	Time	LocalMemSize	VGPRs	SGPRs	ScratchRegs	FCStacks	KernelOccupancy	ALUBusy (%)
1	applySources k1 BeaverCreek1	1	604	269	{ 64	64	1}	{ 64	4	1}	0.03500	0	7	NA	0	0	<u>100</u>	23.35
2	applyBuoyancy k2 BeaverCreek1	2	604	270	{ 64	64	64}	{ 64	4	1}	0.81560	0	4	NA	0	0	<u>100</u>	22.28
3	calculateCurlU k3 BeaverCreek1	3	604	271	{ 64	64	64}	{ 64	4	1}	1.20496	0	9	NA	0	0	<u>75</u>	15.54
4	applyVorticity k4 BeaverCreek1	4	604	272	{ 64	64	64}	{ 64	4	1}	1.72332	0	5	NA	0	2	<u>100</u>	85.82
5	advectFieldVelocity K5 BeaverCreek1	5	604	273	{ 64	64	64}	{ 64	4	1}	0.80228	0	15	NA	0	0	<u>50</u>	54.60
6	applyVelocityBoundaryCondition k6 BeaverCreek1	6	604	274	{ 64	64	64}	{ 64	4	1}	0.97776	0	4	NA	0	0	<u>100</u>	12.85
7	computeFieldPressurePrep k7 BeaverCreek1	7	604	275	{ 64	64	64}	{ 64	4	1}	1.03408	0	9	NA	0	0	<u>75</u>	24.89
8	computeFieldPressureIter k8 BeaverCreek1	8	604	276	{ 64	64	64}	{ 64	4	1}	0.37424	0	7	NA	0	0	100	76.58

At the top of the panel, there is a ""Show Zero Columns"" checkbox. When checked, the session table shows all columns. When unchecked, any column that has a zero or empty value for every row is hidden.

The first several columns in the session are always displayed, even if no performance counters are selected for the profile. A description of these columns for $OpenCL^{\text{TM}}$ applications is given in the following table.

Name	
Method	The kernel name (appended withk[Kerne
ExecutionOrder	The order of execution for the kernel dispat
ThreadID	The thread ID of the host thread that made
CallIndex	The call index of the OpenCL [™] API call tha [™]
GlobalWorkSize	The global work-item size of the kernel.
WorkGroupSize	The work-group size of the kernel.
Time	The time spent (in milliseconds) executing t
LocalMemSize	The amount of local memory (LDS for GPU)
VGPRs	The number of general-purpose vector regi
SGPRs	The number of general-purpose scalar regis
ScratchRegs	The number of scratch registers used by the bottleneck. To reduce this number, reduce t
KernelOccupancy	The kernel occupancy (valid only for GPU d as a percentage of the theoretical maximun

The following table gives a description of these columns for an HSA application.

Name	
Method	The kernel name (appended with the Device
ExecutionOrder	The order of execution for the kernel dispat
ThreadID	The thread ID of the host thread that made
GlobalWorkSize	The global work-item size of the kernel.
WorkGroupSize	The work-group size of the kernel.
LocalMemSize	The amount of local memory (LDS) in bytes
VGPRs	The number of general-purpose vector regi
SGPRs	The number of general-purpose scalar regis
KernelOccupancy	The kernel occupancy. This is an estimate o theoretical maximum number of wavefronts

The following table gives a description of these columns for a DirectCompute application.

Name	Description
Identifier	The kernel name (appended with a pointer vainstance) or the data transfer operation name
ExecutionOrder	The order of execution for the kernel and dat program.
ThreadGroup	The thread group size of the kernel.
WorkGroupSize	The work-group size of the kernel.
Time	For a kernel dispatch operation: time spent (i kernel; does not include the kernel setup time time spent (in milliseconds) transferring data

The <u>GPU Profile: Performance Counters page</u> of the <u>GPU</u> <u>Profiling Project Settings</u> contains the description of the performance counters. This description is also shown if you hover the mouse cursor over the counter name in the Session panel.

To show the source, IL, or ISA code of an OpenCLTM kernel, or the DXASM code of a DirectCompute kernel, click on the kernel name in the first column to open the **<u>GPU Profiler Code</u>** <u>**Viewer**</u>.

For OpenCLTM applications, if a kernel is run on a CPU device, only the global work size, work group size, local memory, and the execution time for the kernel is available.

Using the performance counters lets you:

- Find the number of resources (general-purpose registers, local memory size, and flow control stack size) allocated for the kernel. These resources affect the possible number of in-flight wavefronts in the GPU. A higher number can hide data latency better.
- Determine the number of ALU, global, and local memory instructions executed by the GPU.

- Determine the number of bytes fetched from, and written to, the global memory.
- Determine the use of the SIMD engines and memory units in the system.
- View the efficiency of the shader compiler in packing ALU instructions into the VLIW instructions used by AMD GPUs.
- View any local memory (local data share LDS) bank conflicts.
- View <u>Kernel occupancy percentage</u>, which estimates the number of in-flight wavefronts on a compute unit as a percentage of the theoretical maximum number of wavefronts that the compute unit can support.

To view more information about the <u>kernel occupancy</u> figure for an OpenCL^m kernel, click on the percentage value in the **Kernel Occupancy** column to open the <u>GPU Profiler Kernel</u> <u>Occupancy Viewer</u>.

Note: Special case when other workloads are using the GPU while profiling

When collecting performance counters, it is strongly recommended that no other workloads (i.e. graphics workloads) are running on the GPU. Performance counters on AMD Radeon[™] GPUs are global in nature, meaning that graphics workloads running on the GPU concurrently with a compute workload that is being profiled can affect the counter values reported. It is recommended that all other applications are closed before profiling. Note: The Windows user interface itself uses the GPU for rendering and it may not be possible to disable this. Because of this, there may be some rare occurrences where the counters for a particular kernel dispatch may be incorrect.

Note: Special case when an OpenCL^m kernel uses *printf*

When profiling an OpenCLTM kernel that contains one or more *printf* calls, the Performance Counter results will show values as if the kernel was dispatched with a single wavefront (regardless of how many actual wavefronts are launched). This is due to the way *printf* is implemented in the OpenCL

runtime. When a kernel contains *printf*, internally, the runtime dispatches each wavefront separately. It is recommended that you remove all *printf* statements from a kernel before you attempt to profile it.

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Profiler > Using the GPU Profiler > Application Timeline Trace Summary Pages

Application Timeline Trace Summary Pages

The GPU Profiler Summary Pages show the statistics for your OpenCL[™] or HSA application. They can provide you with a general idea of the location of the application's bottlenecks. They also provide information such as the number of buffers and number of images created on each context (for OpenCL[™]), the most expensive kernel call, etc. One Summary Page, the "Warning(s)/Error(s)" page, shows the result of a rule-based analysis of the API trace and timeline data. You can sort each column in a summary page by clicking the table header. You also can rearrange the columns by dragging them to a new location. By default, the Summary Pages are generated when performing an Application Trace profile from CodeXL. You can control whether the Summary Pages are generated by changing the settings on the **Application Timeline Trace page**. To generate summary pages from the command line, see **Using the Command Line Interface**. You can find summary pages under the same directory as the .atp file. You can view each summary page in your default web browser because all summary pages are in html format.

Context ID		# of Images	# of Kernel Dispatch - BeaverCreek	Total Kernel Time(ms) - BeaverCreek	# of Memory Transfer	Total Memory Time(ms)	# of Read	Total Read Time(ms)	Size of Read	# of Write	Total Write Time(ms)	Size of Write	# of Map	Total Map Time(ms)	Size of Map	# of Copy	Total Copy Time(ms)	Size of Copy
0	14	0	8910	2955.54326	492	812.61090	0	0	0 Byte	330	553.07370	11.06 MB	0	0	0 Byte	162	259.53720	648.00 MB
Total	14	0	8910	2955.54326	492	812.61090	0	0	0 Byte	330	553.07370	11.06 MB	0	0	0 Byte	162	259.53720	648.00 MB

API Summary Page

The API Summary page shows statistics for all OpenCL^m or HSA API calls made by the application. This page can help to identify any API hotspots.

API Name	# of Calls	Total Time(ms)	Ave Time(ms)
clSetKernelArg	60884	106.42239	0.00175
clGetKernelInfo	4332	8.29252	0.00191
clEnqueueNDRangeKernel	4332	178.09443	0.04111
clReleaseMemObject	4265	537.25810	0.12597
clCreateBuffer	4265	13.41322	0.00314
clEnqueueReadBuffer	938	2403.70320	2.56258
clEnqueueWriteBuffer	231	318.01591	1.37669
clEnqueueWriteBufferRect	63	1286.34994	20.41825
clEnqueueReadBufferRect	63	4288.33441	68.06880
clReleaseKernel	13	0.03764	0.00290

clCreateKernel	13	0.06600	0.00508
clGetDeviceInfo	11	0.03422	0.00311
clGetPlatformInfo	5	0.02396	0.00479
clGetContextInfo	4	0.02396	0.00599
clReleaseContext	2	0.00831	0.00415
clCreateContextFromType	2	0.06844	0.03422
clReleaseProgram	1	2.08413	2.08413
clReleaseCommandQueue	1	97.29672	97.29672
clFinish	1	0.06307	0.06307
clCreateProgramWithBinary	1	0.58276	0.58276
clCreateCommandQueue	1	227.11189	227.11189
clBuildProgram	1	1492.02943	1492.02943

OpenCL[™] Context Summary Page

The Context summary page shows the statistics for all the OpenCLTM kernel dispatch and data transfer operations for each context. It also shows the number of buffers and images created for each context.

Context ID	# of Buffers	# of Images	# of Kernel Dispatch - CPU_Device	Total Kernel Time(ms) - CPU_Device	# of Kernel Dispatch - Juniper	Total Kernel Time(ms) - Juniper
0	2	0	1	69.07170	1	4.10271
1	2	0	1	35.53000	NA	NA
2	2	0	NA	NA	1	3.41856
3	2	0	1	35.73143	NA	NA
4	2	0	NA	NA	1	1.38896
Total	10	0	3	140.33313	3	8.91023

Kernel Summary Page

The Kernel summary page shows statistics for all the kernels that are dispatched by the application.

Kernel Name	Device Name		Total Time(ms)	Avg Time(ms)	Max Time(ms)	Mi Tir
multiDeviceKernel	CPU_Device	3	140.33313	46.77771	69.07170	35.
multiDeviceKernel	Juniper	3	8.91023	2.97008	4.10271	1.3

OpenCL™Top 10 Data Transfer Summary Page

The Top 10 Data transfer summary page shows a sorted list of the ten most time-consuming OpenCL[™] data transfers operations. Clicking on a hyperlink takes you to the corresponding item in the Timeline view.

Since data transfer operations can have a great impact on application performance, ensuring that kernel execution operations and data transfer operations overlap can lead to better overall performance.

Command Type	Context ID	Command Queue ID	Duration(ms)	Transfer Size	Transfer Rate(MB/s)
WRITE_BUFFER	4	5	1.45957	256.00 KB	171.284
WRITE_BUFFER	0	1	0.94618	256.00 KB	264.220
WRITE_BUFFER	2	3	0.68449	256.00 KB	365.237
WRITE_BUFFER	1	2	0.21951	256.00 KB	1138.895
WRITE_BUFFER	3	4	0.17307	256.00 KB	1444.527
WRITE_BUFFER	0	0	0.13885	256.00 KB	1800.569

Top 10 Kernel Summary Page

The Top 10 kernel summary page shows a sorted list of the 10

most time-consuming kernel execution operations. Clicking on a hyperlink takes you to the corresponding item in Timeline view.

Kernel Name	Context ID	Command Queue ID	Device Name	Duration(ms)	Global Work Size
<u>multiDeviceKernel</u>	0	0	CPU_Device	69.07170	{65536]
multiDeviceKernel	3	4	CPU_Device	35.73143	{65536]
multiDeviceKernel	1	2	CPU_Device	35.53000	{65536]
multiDeviceKernel	0	1	Juniper	4.10271	{65536]
multiDeviceKernel	2	3	Juniper	3.41856	{65536]
multiDeviceKernel	4	5	Juniper	1.38896	{65536]

Warning(s)/Error(s) Page

The Warning(s)/Error(s) Page shows potential problems in your OpenCL[™] or HSA application. It can detect unreleased resources, API failures, and it can provide suggestions for better performance. Clicking on a hyperlink takes you to the corresponding API.

Index	Call Index	Thread ID	Туре	Message
0	542	2268	Warning	<u>Memory leak detected [Ref =</u> <u>1, Handle = 0x0B1730B0]:</u> <u>Object created by</u> <u>clEnqueueNDRangeKernel</u>
216	208	2268	Best Practices	<u>clEnqueueNDRangeKernel:</u> <u>Work-group size is too small -</u> [1,1,1]. Recommended work- group size is a multiple of 64.
270	319	2268	Best Practices	<u>clEnqueueNDRangeKernel:</u> <u>Global work size is too small -</u> [111], resulting in low GPU <u>utilization.</u>
144	482	1932	Error	<u>clEnqueueNDRangeKernel</u> <u>returns</u> <u>CL_INVALID_KERNEL_ARGS.</u>

From these summary pages, it is possible to determine whether an OpenCL[™] application is bound by kernel execution or data transfer (Context Summary page). If the application is bound by kernel execution, you can determine which device is the bottleneck. From the Kernel Summary page, you can find the name of the kernel with the highest total execution time. From the Top 10 Kernel Summary page, you can find the individual kernel instance with the highest execution time. If the kernel execution on a GPU device is the bottleneck, the GPU performance counters can then be used to investigate the bottleneck inside the kernel.

If the application is bound by the data transfers, it is possible to determine the most expensive data transfer type (read, write, copy or map) in the application from the Context Summary page. You can then investigate whether you can minimize this type of data transfer by modifying the algorithm if necessary. With help from the Timeline View, you can investigate whether data transfers have been executed in the most efficient way (concurrently with a kernel execution).

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Profiler > Using the GPU Profiler > GPU Profiler Code Viewer

GPU Profiler Code Viewer

The Code Viewer appears when you click on the name of a kernel in the first column of the **<u>GPU Profiler Performance</u>** <u>**Counters Session**</u> panel.

SA				•
00 ALU:	ADDR(32) CNT(85) KCACH	HE0(CB0:0-15) KCACHE1(CB1:0-15)	
0	x: MOV	R8.x,	0.0f	
	t: MULLO_INT	,	R1.y, KC0[1].y	
1	z: ADD_INT	,	R0.y, PS0	
	t: MULLO_INT	,	R1.x, KC0[1].x	=
2	x: ADD_INT	,	PV1.z, KC0[6].y	
	w: ADD_INT	,	R1.x, KC0[1].x PV1.z, KC0[6].y R0.x, PS1 R1.z, KC0[1].z	
	t: MULLO_INT	,	R1.z, KC0[1].z	
3	x: ADD_INT	,	PV2.w, KC0[6].x R0.z, PS2	
	y: ADD_INT	,	R0.z, PS2	
	z: LSHL	,	PV2.x, 6	
	t: I_TO_F			
4	x: SETGE_DX1		PS3, 1.0f	
	y: SETNE_DX1		PS3, PS3	
	z: ADD_INT		PV3.y, KC0[6].z	
			PV3.x, PV3.z	
	t: I_TO_F			
5			PV4.y, PV4.x	
	y: LSHL		PV4.z, 12	
<	z: SETGT DX1	0 T1.z.	PS4. (0x42780000. 62.0f).v	

For OpenCL^m kernels, this panel shows the generated ISA or IL/HSAIL code of the kernel. It also shows the CL source code of the kernel, if the kernel source is available from the OpenCL^m runtime.

For DirectCompute kernels, this panel shows the DXASM code of the kernel.

For OpenCL[™] kernels, you can select different modes in the combo box at the top of the panel to switch between displaying the IL code, ISA code, and CL source code.

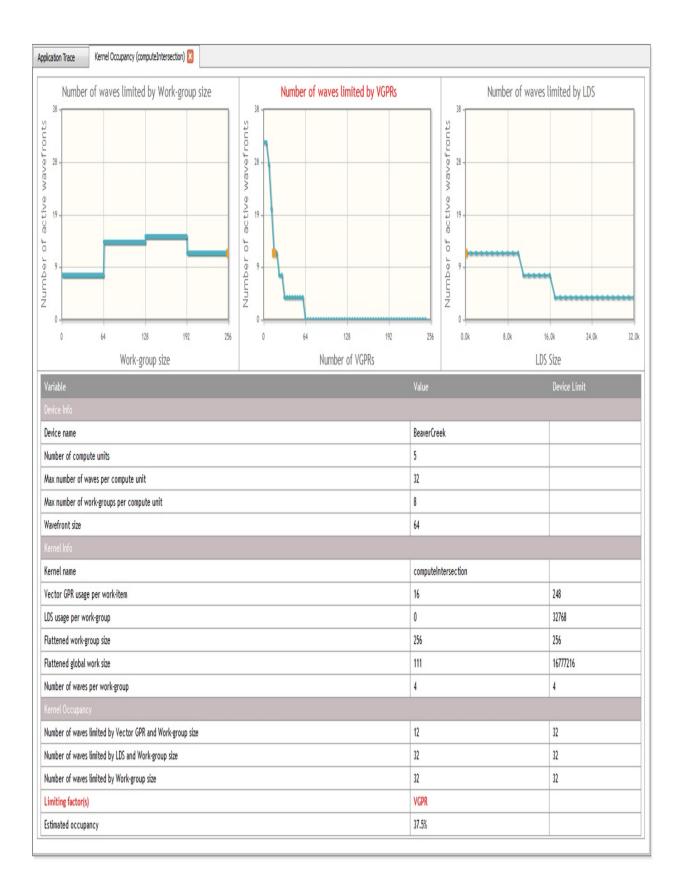
 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Profiler > Using the GPU Profiler > GPU Profiler Kernel Occupancy Viewer

GPU Profiler Kernel Occupancy Viewer



There are two ways to open the Kernel Occupancy panel:

- Click on the kernel occupancy percentage in the Kernel Occupancy column of the GPU Profiler Performance Counters Session panel (OpenCL[™] and HSA). or
- Click on the <u>kernel occupancy percentage</u> in the Kernel Occupancy column of the API Trace in the <u>GPU Profiler</u> <u>Profiler Application Trace Session</u> panel (OpenCL[™] only).

For kernels, this panel displays an HTML webpage which provides information about the occupancy of a particular kernel dispatch.

The top part of the page shows four graphs that provide a visual indication of how kernel resources affect the theoretical number of in-flight wavefronts on a compute unit. The graph representing the limiting resource has its title displayed in red text. More than one graph can have a red title if there is more than one limiting resource. In each graph, the actual usage of the particular resource being graphed is highlighted with an orange square. If you hover the mouse over a point in the graph, a popup hint is displayed showing you the current X and Y values at that location.

The first graph, titled **Number of waves limited by Workgroup size**, shows how the number of active wavefronts is affected by the size of the work-group for the dispatched kernel. In the screenshot above, you can see that the highest number of wavefronts is achieved when the work-group size is in the between 64 and 256.

The second graph, titled **Number of waves limited by VGPRs,** shows how the number of active wavefronts is affected by the number of vector GPRs used by the dispatched kernel. In the screenshot above, you can see that as the number of VGPRs used increases, and the number active wavefronts decreases, in steps. Note this graph shows that more than 62 VGPRs can be allocated, even though 62 is the maximum number of VGPRs that can be allocated, since the shader compiler assumes the work-group size is 256 items by default (the largest possible work-group size). For the shader compiler to allocate more than 62 VGPRs, the kernel source code must be marked with the required_work_group_size kernel attribute. This attribute specifies to the shader compiler that the kernel is launched with a work-group size smaller than the maximum, allowing it to allocate more VGPRs. Thus, for X-axis values greater than 62, the VGPR graph shows the theoretical number of wavefronts that can be launched if the kernel specified a smaller work-group size using the attribute.

The third graph, titled **Number of waves limited by SGPRs**, shows how the number of active wavefronts is affected by the number of scalar GPRs used by the dispatched kernel. In the above screenshot, you can see that as the number of SGPRs used increases, the number active wavefronts decreases in steps.

The fourth graph, titled **Number of waves limited by LDS**, shows how the number of active wavefronts is affected by the amount of LDS used by the dispatched kernel. In the above screenshot, you can see that as the amount of LDS used increases, the number active wavefronts decreases in steps.

A table, below the four graphs, provides information about the device, the kernel, and the kernel occupancy. In the **Kernel Occupancy** section, you can see the limits imposed by each kernel resource, as well as which resource is currently limiting the number of waves for the kernel dispatch. This section also displays the **kernel occupancy** percentage.

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Profiler > Using the GPU Profiler > GPU Profiler Kernel Occupancy

GPU Profiler Kernel Occupancy

This page provides an overview of the kernel occupancy calculation, providing the definition of the parameter and discussing the factors influencing the value and its interpretation.

Kernel occupancy is a measure of the use of the resources of a compute unit on a GPU, the use being measured by the number of in-flight wavefronts, for a given kernel, relative to the number of wavefronts that can be launched given the ideal kernel dispatch configuration (dependent on the work-group size and resource use in the kernel).

The number of wavefronts that are scheduled when a kernel is dispatched is constrained by three significant factors:

- the number of general purpose registers (GPR) required by each work-item,
- the amount of shared memory (LDS for local data store) used by each work-group, and
- the configuration of the work-group (the work-group size).

The basic definition of the occupancy (O) is given by:

$$O = \frac{N_w^A}{N_w^T}$$

where N_W^A is the number of in-flight wavefronts on the compute unit, and N_W^T is the theoretical number of wavefronts that the compute unit can execute concurrently.

The first constraint is that work that is assigned to a compute unit is scheduled as groups of individual work-items, called wavefronts, which have a fixed size defined by the hardware. The characteristic of a wavefront is that each work-item executes in step with the other work-items in the wavefront. The number of work-items that can be executed on a compute unit must be a multiple of a wavefront. In an ideal situation, the number of wavefronts that can be scheduled corresponds to the maximum number of wavefronts supported by the compute unit.

However, because there are resources that are shared among work-groups, which is the basic unit of processing on the compute unit, wavefronts are scheduled as part of a workgroup. A work-group consists of a collection of work-items that make use of a common block of local data storage (LDS) that is shared among the members of the work-group, as well as registers. Each work-group consists of one or more wavefronts. Thus, the total number of wavefronts that can be launched on a compute unit is also constrained by the number of work-groups as this must correspond to an integral number of workgroups, even if the compute unit has capacity for additional wavefronts. In the ideal situation, the number of wavefronts that can be launched is an integral multiple of the number of wavefronts per work-group, which means that the maximum number of wavefronts the GPU is capable of allocating, can be achieved. When this is not the case, changing the size of the work-items in the work-group can change the number of wavefronts in the work-group.

Kernel Occupancy for AMD Radeon[™] HD 5000/6000 Series Based on VLIW5/VLIW4 Architecture

1. LDS limits on the number of in-flight

In the case that the LDS is the only constraint on the number of in-flight wavefronts, the compute unit can support the launch of a number of in-flight work-groups given by:

$$WG_{max} = \frac{LDS_{CU}}{LDS_{WG}}$$

where WG_{max} is the maximum number of work-groups on a compute unit, LDS_{CU} is the shared memory available on the compute unit, and LDS_{wg} is the shared memory required by the work-group (based on the resources required by the kernel). The corresponding number of wavefronts is given as:

 $WF_{max} = WG_{max} * WF_{WG}$

where WF_{max} is the maximum number of wavefronts, WG_{max} is the maximum number of work-groups, and WF_{WG} is the number of wavefronts in a work-group.

There is also another constraint whereby a compute unit can only support a fixed number of work-groups, a hard limit of $WG_{max}=8$ (denoted by WG_{max}^{CU}). This also limits the effectiveness of reducing the work-group size excessively, as the number of wavefronts is also limited by the maximum workgroup size. Currently, the maximum work-group size is 256 work-items, which means that the maximum number of wavefronts is 4 when the wavefront size is 64 (and 8 when the wavefront size is 32).

Thus, when the only limit to the number of wavefronts on a compute unit is set by the LDS usage (for a given kernel), then the maximum number of wavefronts, (LDS-limited) is given by:

 $WF_{LDS}^{max} = \min(WG_{max}^{CU} * WF_{WG}, WG_{max} * WF_{WG})$

2. GPR limits on the number of in-flight wavefronts>

Another limit on the number of in-flight wavefronts is the number of general-purpose registers (GPRs). Each compute unit has 16384 registers. These are divided among the workitems in a wavefront. Thus, the number of registers per workitem limits the number of wavefronts that can be launched. This can be expressed as:

$$W\!F_{\textit{GPR}} = \frac{N_{\textit{reg}}^{\max}}{N_{\textit{reg}}^{\textit{max}}}$$

where N_{reg} is the number of registers per work-item; the superscripts max and used refer to the maximum number of registers per thread and the actual number of registers used. The number of in-flight wavefronts being constrained by the work-group granularity, the number of GPR-limited wavefronts is given by:

 $WF_{GPR}^{max} = floor \left(\frac{WF_{GPR}}{WF_{WG}}\right) * WF_{WG}$

3. Other constraints

Another limit on the number of in-flight wavefronts is the FCStack; however, this is really an insignificant constraint, so this is not considered here.

The final factor in the occupancy is the work-group size, as briefly discussed above. If there are no other constraints on the number of wavefronts on the compute unit, the maximum number of wavefronts is given by:

 $WF_{WG}^{\max} = \min\left(floor\left(\frac{WF_{\max}^{CU}}{WF_{wg}}\right), WG_{\max}^{CU}\right) * WF_{WG}$

where WF_{max}^{CU} is the maximum number of wavefronts on the compute unit and WF_{WG}^{max} is the maximum number of wavefronts on a compute unit when there are no other constraints than the work-group size.

This equation shows that having a workgroup size where the number of wavefronts divides the maximum number of wavefronts on the compute unit evenly generally yields the greatest number of in-flight wavefronts, while at the same time indicating that making the work-group size too small yields a reduced number of wavefronts. For example, setting a workgroup consisting of only 1 wavefront yields only 8 in-flight wavefronts, whereas (for example, given a maximum number of wavefronts on the compute unit of 32), a work-group of 2 wavefronts will yield 16 wavefronts. Furthermore, having a single wavefront per work-group doubles the LDS usage relative to having 2 wavefronts per work-group as the LDS is only shared among the wavefronts in a same work-group (but not between work-groups).

Given these constraints, the maximum number of in-flight wavefronts is given by:

 $N_{W}^{A} = \min \left\{ WF_{LDS}^{\max}, \min \left[WF_{WG}^{\max}, WF_{GPR}^{\max} \right] \right\}$

Thus, the occupancy, O, is given by:

 $O = \frac{\min\left\{WF_{LDS}^{\max}, \min\left[WF_{WG}^{\max}, WF_{GPR}^{\max}\right]\right\}}{N_{w}^{T}}$

The occupancy shown here is the estimated occupancy on a single compute unit. It is independent of the work-loads on the other compute units on the GPU because the occupancy is only really meaningful if there are sufficient work-items to require all the resources of at least one compute unit (and even then, ideally, there should be a sufficient work-load to ensure that more than one compute unit is needed to execute the work in order to gain the benefits of parallel operations). Higher occupancy allows for increased global memory latency hiding as it allows wavefronts to be swapped when there are global memory accesses. However, once there is a sufficient number of wavefronts on the compute unit to hide any global memory accesses, increasing occupancy may not increase performance.

Kernel Occupancy for AMD Radeon[™] HD 7000 Series or Newer, Based on Graphics Core Next Architecture

There are a number of significant differences from the previous occupancy calculation due to the different architecture. In the Graphics Core Next architecture, each compute unit is actually made up of four SIMDs. While some features, such as the GPR, are still computed on the basis of individual SIMDs, these must be scaled to the whole compute unit. On the other hand, workgroup limits must be computed over the whole compute unit. These are detailed below.

The first limit to the number of active wavefronts on the compute unit is the work-group size. Each Compute unit (CU), has up to 40 slots for wavefronts. If each work-group is exactly one wavefront, then the maximum number of wavefronts is: $WF_{max} = 40$

Otherwise, if there is more than one wavefront (WF) per workgroup (WG), there is an upper limit of 16 work-groups (WG) per compute unit (CU). Then, the maximum number of wavefronts on the compute unit is given by:

 $WF_{WG}^{max} = \min(16 * WF_{WG}, WF_{max})$

where WF_{WG} is the number of wavefronts per work group.

The second limit on the number of active wavefronts is the number of VGPR per SIMD.

 $WF_{VGPR}^{max} = floor(\frac{VGPR_{max}}{VGPR_{used}})$

Where $VGPR_{max}$ is maximum number of registers per work-item and $VGPR_{used}$ is the actual number of registers used per work-item. However, we are interested in the total number of wavefronts per CU, so we have to scale this value by the number of CU.

 $WF_{VGPR}^{max} = WF_{VGPR}^{max} * SIMD_{PER_{CU}}$

At the same time, the number of wavefronts cannot exceed $\mathrm{WF}_{\mathrm{max}}\text{, so}$

 $WF_{VGPR}^{max} = \min(WF_{VGPR}^{max}, WF_{max})$

However, the wavefronts are constrained by work-group granularity, so the maximum number of wavefronts limited by the VGPR is given by

 $WF_{VGPR}^{max} = \operatorname{floor}\left(\frac{WF_{VGPR}^{max}}{WF_{WG}}\right) * WF_{WG}$

The third limit on the number of active wavefronts is the number of SGPR. Similar to VGPR, SGPR is calculated by

$$WF_{SGPR}^{max} = floor \left(\frac{min(floor(\frac{SGPR_{max}}{SGPR_{used}}) * SIMD_{PER_{CU}}, WF_{max})}{WF_{WG}} \right) * WF_{WG}$$

The final limit on the number of active wavefronts is the LDS. The LDS limited number of wavefronts is given by:

$$WG_{max} = floor(\frac{LDS_{max}}{LDS_{used}})$$

where WG_{max} is the maximum number of work-groups determined by the LDS. Then, the maximum number of wavefronts is given by:

 $WF_{LDS}^{max} = WG_{max} * WF_{WG}$

Thus, the occupancy, O, is given by:

 $O = \frac{\min\{WF_{LDS}^{max}, WF_{SGPR}^{max}, WF_{VGPR}^{max}, WF_{WG}^{max}\}}{WF_{max}}$

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Profiler > Using the GPU Profiler > GPU Profiling Project Settings

GPU Profiling Project Settings

These Project Settings pages let you configure various aspects of the GPU Profiler for the active project.

The following pages contain the settings that can be configured:

Application Timeline Trace page

GPU Profile: Performance Counters page

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.

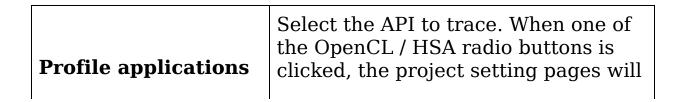


Application Timeline Trace page

This page lets you configure the behavior of the Profiler when it performs an application timeline trace.

Application Timeline Trace settings page

CodeXL Project Settings			Х
General Debug Profile Profile Application Timeline Trace GPU Profile: Perf. Counters Power Profile Frame Analysis Analyze Nernel/Shader Build Options HLSL Build Options	Application Timeline Trace Profile applications that use: OpenCL HSA Enable navigation to source code (high overhead) Write trace data in intervals during program execution (ms): 100 Maximum number of APIs to trace: 200000 OpenCL Trace Options Always show API error codes Collapse consecutive identical dGetEventInfo calls Generate occupancy information for each OpenCL or HSA kernel profiled API Trace Options Mil OpenCL Rules Warnings Best Practices Frors 	 API to trace API to trace ✓ AII OpenCL APIs > ✓ CLObjectCreate > ✓ CLObjectRelease > ✓ CLObjectRelease > ✓ CLObjectRelease > ✓ CLObjectRelease > ✓ DirectXInterOp > ✓ EnqueueDataTransfer > ✓ EnqueueOther > ✓ EnqueueOther > ✓ OpenGLInterOp > ✓ OpenGLInterOp > ✓ Other > ✓ OpenGLInterOp > ✓ SetCallback > ✓ Synchronization 	
Restore Default Settings		OK Cancel	



that use	display the options relevant to the selected API.
Enable navigation to source code (high overhead)	When checked, the Profiler generates a symbol information file from an application's debugging information (the .pdb file on Windows), containing one entry for each called OpenCL [™] API. This symbol information file lets you navigate from an item in the API Trace in the <u>Application Timeline Trace</u> <u>Session</u> panel to the source location of the API call.
Write trace data in intervals during program execution (ms)	When selected, the Profiler periodically writes all queued trace data to disk during program execution. The interval (in milliseconds) at which to write trace data is specified using the value following the checkbox. When checked, in addition to writing data periodically, the Profiler also writes all queued trace data when the clReleaseContext OpenCL [™] API is called. However, if an application does not call clReleaseContext , or if it calls any OpenCL [™] APIs after the final clReleaseContext call, then it is possible that not all trace data is written to the disk. When unchecked, all trace data is written to disk when the application terminates. On Linux, this is the default (and only supported) mode for writing trace data. Thus on Linux, the UI lets you

	specify the interval but does not let you enable or disable writing the data in intervals.
Maximum number of APIs to trace	This controls how many APIs are traced over an application's lifetime. The default number of APIs to trace is 1 million. Limiting the number of APIs traced helps to prevent running out of memory while profiling. After the limit is reached, no additional APIs is traced, and the trace results do not include any additional information. Because of this, any information provided in the GPU Profiler Summary Pages might not be correct, as a complete trace is required to provide a fully-accurate application summary.
Always show API error codes	When checked, the Profiler reports the return codes for all OpenCL [™] API calls. Some OpenCL [™] API functions return an error code through a passed-in parameter. If the host application passes in NULL for that parameter, then the OpenCL [™] runtime does not report an error code. The Profiler substitutes a non- null parameter in this case, and the API Trace can show the return code.
	Some OpenCL [™] applications wait for certain Enqueue API calls to complete by continuously checking the status of the event returned by the Enqueue API. These applications do this by calling clGetEventInfo

Collapse consecutive identical clGetEventInfo calls	within a loop until the event status reaches a certain state (typically CL_COMPLETE). For these applications, the timeline and API trace can contain thousands of clGetEventInfo calls, making it difficult to easily analyze the timeline and trace data. To make analysis easier, the Profiler can collapse consecutive clGetEventInfo calls that have the same parameters and return values into a single entry in the timeline and API trace.
Generate occupancy	When checked, the Profiler generates
information for each	<u>kernel occupancy</u> data for each
OpenCL kernel	OpenCL [™] kernel dispatched to a
profiled	GPU device.

Generate summary pages

When checked, the Profiler automatically generates **GPU Profiler Summary Pages** using the API trace and timeline data. You can further configure the summary pages by selecting rules to be used when generating the Warning(s)/Error(s) Summary page. The following table shows the currently supported rules.

Rule	I
Detect resource leaks	Tracks the reference count for all O objects not released
Detect deprecated API calls	Detects calls to OpenCL [™] API funct versions of OpenCL [™]

Detect unnecessary blocking writes	Detects unnecessary blocking write
Detect non-optimized work size	Detects clEnqueueNDRangeKernel size that is non-optimal for AMD Ha
Detect non-optimized data transfer	 Detects non-Fusion APU access to Detects host-visible Device Memory
Detect redundant synchronization	Detects redundant synchronization
Detect failed API calls	Detects OpenCL [™] API calls that do Detects HSA API calls that do not r€ Some of the return codes from Open Always show API error codes opti

APIs to trace When checked, you can tell the Profiler which APIs you want traced. By limiting the APIs to trace, you can focus attention on particular APIs when analyzing trace data while also reducing the overhead of performing a trace. Because a full trace is required in order to generate the Summary pages, this option is mutually exclusive with the **Generate summary pages** option. Use the treeview below the option to select the APIs for the Profiler to trace.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.



GPU Profile: Performance Counters page

This page lets you configure the behavior of the Profiler when it collects performance counters.

🛚 CodeXL Project Settings		Х
General Debug Profile > CPU Profile Application Timeline Trace GPU Profile: Perf. Counters Power Profile Frame Analysis	General Collect counters for: OpenCL HSA Performance Counters Selection Collecting Multiple performance counters may require executing the kernel more than once. If multiple passes are required, the profiler stores the OpenCL real/write buffers prior to kernel execution and restores them afterwards. Image: Selection V General Image: Wavefronts Value Value Selection Image: Selection Selection Selection Selection and restores them afterwards. Image: Selection Selection Selection and restores them afterwards. Selection Se	
Restore Default Settings	Profile only these kernels (names are case-sensitive, separated by semi-colons). Leave empty to profile all kernels.	ancel

Settings

- Measure kernel execution time when checked, requires an additional pass during collection. (only applicable for OpenCL)
- Generate occupancy information for each OpenCL[™] or HSA kernel profiled When checked, the Profiler generates <u>kernel occupancy</u> data for each OpenCL[™] kernel dispatched to a GPU device.
- **Profile specific kernels** Profile only kernels that their names are specified.
- Counter selection TreeView This treeview displays the available GPU performance counters that can be enabled for a profile session. The performance counters are grouped by counter type. The counters shown depend on the type of GPU installed on the system. If the system has multiple GPU devices from multiple hardware families, the tree contains a top-level node for each available hardware family. For instance, if a system has both an AMD Radeon[™] HD 7000 series GPU device (one based on Graphics Core Next Architecture) and an AMD Radeon[™] HD 5000 series device, then the counter selection treeview includes counters supported by each device (see screenshot below).
- Some counter selection combinations require multi-pass collection. When profiling using multiple passes, any OpenCL kernels that use shared virtual memory or pipes as arguments will not be profiled.
- When more than one pass is required, the number of required passes will be displayed next to the device name.
- To load and save the counter selections to a file, click on the **Load Selection** and **Save Selection** buttons.

Below is a list and brief description of available counters. You also can use the cursor to hover over the counter names in the treeview to view the descriptions.

The full set of counters for AMD Radeon[™] HD 7000 series GPU devices or newer (based on Graphics Core

Next Architecture) are described in the following table.

Name	Descripti
Wavefronts	Total wavefronts.
VALUInsts	The average number of vector ALU instruct (affected by flow control).
SALUInsts	The average number of scalar ALU instructi (affected by flow control).
VFetchInsts	The average number of vector fetch instruction executed per work-item (affected by flow cc
SFetchInsts	The average number of scalar fetch instruct executed per work-item (affected by flow cc
VWriteInsts	The average number of vector write instruc per work-item (affected by flow control).
FlatVMemInsts	The average number of FLAT instructions the memory executed per work item (affected b instructions that read from or write to scrat
GDSInsts	The average number of GDS read or GDS w item (affected by flow control).
VALUUtilization	The percentage of active vector ALU thread mean either more thread divergence in a wanot a multiple of 64. Value range: 0% (bad), divergence).
VALUBusy	The percentage of GPUTime vector ALU ins range: 0% (bad) to 100% (optimal).
SALUBusy	The percentage of GPUTime scalar ALU instrange: 0% (bad) to 100% (optimal).
LDSInsts	The average number of LDS read or LDS wi item (affected by flow control).
FlatLDSInsts	The average number of FLAT instructions the per work item (affected by flow control).
LDSBankConflict	The percentage of GPUTime LDS is stalled

	(optimal) to 100% (bad).
FetchSize	The total kilobytes fetched from the video n extra fetches and any cache or memory effe
WriteSize	The total kilobytes written to the video men fetches and any cache or memory effects ta
CacheHit	The percentage of fetch, write, atomic, and cache. Value range: 0% (no hit) to 100% (op
MemUnitBusy	The percentage of GPUTime the memory un stall time (MemUnitStalled). This is measur and any cache or memory effects taken into (fetch-bound).
MemUnitStalled	The percentage of GPUTime the memory un number or size of fetches and writes if poss 100% (bad).
WriteUnitStalled	The percentage of GPUTime the Write unit (bad).

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Profiler > Using the GPU Profiler > Description of Output Files

Description of Output Files

SESSION_NAME.csv

This comma-delimited file is generated when a profile collects performance counters.

The file starts with a file header section (in comments) that indicates the Profiler version number and information about the application that was profiled. Following the file header is a line containing the list of the column headers shown in the **GPU Profiler Performance Counters Session** panel. Most items in this row represent the performance counters that were collected.

Each additional line contains data collected by the Profiler. There will be one line for each kernel dispatched by the profiled application.

SESSION_NAME.atp

This file is generated when performing a profile that collects an application timeline trace. The file starts with a file header section which contains the trace file version number, the Profiler version number, and information about the application that was profiled. Following the file header are several sections: the first section contains the API Trace data for the profile session; the second contains timestamp data for the profile session. For HSA traces that include HSA kernel dispatches, there will be a section containing the kernel dispatch timestamp data. If the option to Enable navigation to source code is checked on the **Application Timeline Trace page**, there will be a section containing the source code

information for the profile section.

The API Trace section contains one or more **thread blocks**.

An API Trace **thread block** consists of the following.

- · A line giving the thread ID.
- A line giving the number of APIs for that thread, followed by a line for each API.

Each API is listed in the format: **ReturnValue = APIName (ParameterList)**.

The **ParameterList** is a semi-colon delimited list of the parameters passed to the API.

The Timestamp section contains one or more **thread blocks**. In the Timestamp section, all time counter data represents CPU-based time expressed in nanoseconds. A Timestamp **thread block** consists of the following.

- A line giving the thread ID.
- A line giving the number of APIs for that thread, followed by an **API line** for each API. An **API line** consists of at least 4 pieces of data:
 - An integer representing the API type.
 - A string showing the API name.
 - The time counter value for the start of the API.
 - The time counter value for the end of the API.

Most OpenCL[™] Enqueue APIs contain the following additional data, appended to the end of the **API line**.

- An integer representing the enqueue command type.
- A string showing the enqueue command name.
- The time counter value for the time the command was queued by the host this corresponds to CL PROFILING COMMAND QUEUED.
- The time counter value for the time the command was submitted by the host to the target device – this corresponds to CL_PROFILING_COMMAND_SUBMIT.
- The time counter value for the time the command started executing on the target device this corresponds to

CL_PROFILING_COMMAND_START.

- The time counter value for the time the command finished executing on the target device – this corresponds to CL_PROFILING_COMMAND_END.
- The unique numerical ID of the queue.
- The handle of the queue.
- The unique numerical ID of the context.
- The handle of the context.
- The device name.

OpenCL[™] Kernel dispatch Enqueue commands contain the following additional data appended to the end of the **API line**.

- The handle of the kernel.
- The name of the kernel.
- The global work size for the kernel one value is given for each work dimension.
- The work-group size for the kernel one value is given for each work dimension.

OpenCL[™] Data transfer Enqueue commands contain the data transfer size appended to the end of the **API line**.

The HSA Kernel Timestamp section contains the following information

- A line giving the number of HSA kernel dispatches, followed by a **Kernel Timestamp line** for each kernel dispatched by the application. A **Kernel Timestamp line** consists of the following pieces of data:
- A string showing the kernel symbol name.
- The handle of the kernel.
- The time counter value for the time the kernel started executing on the device.
- The time counter value for the time the kernel finished executing on the device.
- The name of the agent the where the kernel was dispatched.
- The handle of the agent where the kernel was dispatched.
- The zero-based index of the queue that was used to

dispatch the kernel.

The handle of the queue that was used to dispatch the kernel.

The Source Code section contains one or more **thread blocks**. A Source Code **thread block** consists of the following.

- A line giving the thread ID.
- A line giving the number of APIs for that thread, followed by a **Source Code line** for each API. A **Source Code line** consists of the following 4 pieces of data:
 - A string showing the API name.
 - A string showing the name of the function that called the API (or an address if no debug information was found).
 - An integer representing the line number for the location of the API call.
 - A string showing the name of the file for the location of the API call (this is not shown if no debug information was found).

SESSION_NAME.occupancy

This comma-delimited file is generated when a profile collects kernel occupancy information.

The file starts with a file header section (in comments) that indicates the Profiler version number and information about the application that was profiled. Following the file header is a line containing the list of names of the data used in order to compute kernel occupancy.

Each additional line contains data collected by the Profiler. There will be one line for each kernel dispatched by the profiled application to a GPU device.

 $[\]ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Profiler > Using the GPU Profiler > Description of Configuration Files

Description of Configuration Files

Format of counter configuration file (argument passed to --counterfile)

To specify a set of performance counters to enable when profiling from the command line, pass the name of a configuration file to the **--counterfile** option. You can generate a counter configuration file from within the Visual Studio client by using the "Save Counters" button on the **GPU Profile: Performance Counters page** of the Project Settings dialog. The format of this configuration file is one counter name per line. Counter names are case-sensitive. An example of the contents of this file is given below.

Wavefronts
VALUInsts
SALUInsts
VFetchInsts
SFetchInsts
VWriteInsts
LDSInsts
GDSInsts
VALUUtilization
VALUBusy
SALUBusy
FetchSize
WriteSize
CacheHit
MemUnitBusy
MemUnitStalled
WriteUnitStalled
LDSBankConflict

Format of kernel list configuration file (argument passed to --kernellistfile)

To specify a set of kernels to profile when collecting

performance counters from the command line, pass the name of a configuration file to the **--kernellistfile** option. The format of this configuration file is one kernel name per line. Kernel names are case-sensitive. When specified, any kernels dispatched by the application that are not contained in the kernel list configuration file will not be profiled. An example of the contents of this file is given below.

MatrixMultiplyKernel binarySearch binomial_options

Format of API rules configuration file (argument passed to --apirulesfile)

To specify a set of rules to use when generating the summary pages from a trace file when using the command line, pass the name of a configuration file to the **--apirulesfile** option. The format of this file is one rule per line in the NAME=VALUE format. An example of the contents of this file is given below. Note that the "VALUE" can be either "True" or "False".

APITrace.APIRules.RefTracker=True APITrace.APIRules.BlockingWrite=False APITrace.APIRules.BadWorkGroupSize=True APITrace.APIRules.RetCodeAnalyzer=True APITrace.APIRules.DataTransferAnalyzer=True APITrace.APIRules.SyncAnalyzer=True APITrace.APIRules.DeprecatedFunctionAnalyzer=True

Format of API filter configuration file (argument passed to --apifilterfile)

To ignore a set of APIs when collecting an API trace using the command line, pass the name of a configuration file to the **-apifilterfile** option. The format of this file is one API name per line. An example of the contents of this file for an OpenCLTM is given below.

clGetPlatformIDs clGetPlatformInfo clGetDeviceIDs clGetDeviceInfo clGetContextInfo clGetCommandQueueInfo clGetSupportedImageFormats clGetMemObjectInfo clGetImageInfo clGetSamplerInfo clGetProgramInfo clGetProgramBuildInfo clGetKernelInfo clGetKernelWorkGroupInfo clGetEventInfo clGetEventProfilingInfo

Format of environment variable file (argument passed to --envvarfile)

To specify a set of environment variables to be defined for the application being profiled, pass the name of a configuration file to the **--envvarfile** option. The format of this file is one environment variable per line in the NAME=VALUE format. An example of the contents of this file is given below.

APPLICATION_DATA_DIR=c:\path\to\app\data DEBUG_FLAG=True LOG_FILE=c:\temp\logfile.log

Format of occupancy display configuration file (argument passed to --occupancydisplay)

To generate a Kernel Occupancy HTML display file using the command line, pass the name of a configuration file to the **--occupancydisplay** option. The format of this configuration file is one parameter per line in the NAME=VALUE format. An example of the contents of this file is given below. The "VALUES" are taken from a generated .occupancy file for a particular kernel.

ThreadID=3364 CallIndex=101 KernelName=reduce DeviceName=Capeverde ComputeUnits=10 MaxWavesPerComputeUnit=40 MaxWorkGroupPerComputeUnit=16 MaxVGPRs=256 MaxSGPRs=512 MaxLDS=32768 UsedVGPRs=11 UsedSGPRs=20 UsedLDS=4096 WavefrontSize=64 WorkGroupSize=256 WavesPerWorkGroup=4 MaxWorkGroupSize=256 MaxWavesPerWorkGroup=4 GlobalWorkSize=256 MaxGlobalWorkSize=16777216 WavesLimitedByVGPR=40 WavesLimitedBySGPR=40 WavesLimitedByLDS=32 WavesLimitedByWorkgroup=40 Occupancy=80

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Profiler > Using the GPU Profiler > Using the Command Line Interface

Using the Command Line Interface

- 1. Go to the location of the GPU Profiler binaries. The GPU Profiler binaries are located in either the $\mathbf{x86}$ or $\mathbf{x64}$ ($\mathbf{x86}_6\mathbf{4}$ on Linux) subdirectory under the CodeXL installation directory. Alternatively, you can include the location of the Profiler binaries into the system's path environment variable.
- 2. Run the Profiler using the following instructions.

Usage: CodeXLGpuProfiler <options> InputApplication [InputApplication's command line arguments]

Note: When profiling JOCL (Java OpenCL) application the full path to the java VM must be provided. E.g CodeXLGpuProfiler <options> /usr/bin/java -jar <jar_file>

Note: On some Linux systems which have an older version of the libstdc++ shared object, you may get an error when trying to profile an application from the command line. The error will say "Failed to generate profile result". If you encounter this error, please try using the **CodeXLGpuProfilerRun** shell script to profile. This shell script will first set up the LD_LIBRARY_PATH environment variable to allow the profiler to find the correct version of the libstdc++ shared object. The command-line syntax for the CodeXLGpuProfilerRun shell script is identical to the syntax used for CodeXLGpuProfiler.

General options:

	Start the application with profiling dis amdtStopProfiling and amdtResumePi

-e [envvar] arg	Environment variable that should be c should be in the format NAME=VALU
-E [envvarfile] arg	Path to a file containing a list of environments profiled application. The file should co
-f [fullenv]	The environment variables specified v not specified, then the environment va environment block.
-l [list]	Print the list of valid counter names.
-N [sessionname] arg	Name of the generated session. If not
-o [outputfile] arg	Path to OutputFile. If not provided, th current user's Documents directory; v same location. For Linux, the default
-v [version]	Print the CodeXLGpuProfiler version 1
-w [workingdirectory] arg	Set the working directory. If not provi
-h [help]	Show a help message.

Profile mode options:

-t [apitrace]	Trace OpenCL [™] application and generate CPU and GPU time stamps and detailed API call traces.
-p [perfcounter]	Get the performance counters for each OpenCL [™] or DirectCompute kernel dispatched by the application.
-A [hsatrace]	Trace HSA application and generate CPU and GPU time stamps and detailed API call traces. (Linux only)
	Get the performance counters for

-C [hsapmc]	each HSA kernel dispatched by the application.
-O [occupancy]	Generate kernel occupancy information file (.occupancy).
-P [occupancydisplay] arg	Path to configuration file to use to generate an occupancy display file. Specify the occupancy display file that is to be generated withoutputfile. See below for information about the configuration file format.
-T [tracesummary]	Generate summary page from an input .atp file.

Application Trace mode options (for --apitrace and -hsatrace):

-F [apifilterfile]	Path to the API filter file which contains a list of OpenCL [™] APIs to be filtered out when performing an API trace. See below for information about the API filter file format.
-i [interval] arg (=100)	Timeout interval. Ignored when not performing an API trace and using timeout mode.

-m [timeout]	Flush Trace data periodically, default timeout interval is 100 milliseconds (can be changed with -i option). Ignored when not performing an API trace. (Windows only, this is the default mode for Linux.)
-M [maxapicalls] (=1000000)	Maximum number of API calls.
-n [nocollapse]	Do not collapse consecutive identical clGetEventInfo calls into a single call in the trace output. Ignored when not performing an API trace.
-r [ret]	Always include the OpenCL [™] API return code in API trace, even if client application does not query it. Ignored when not performing an API trace.
-y [sym]	Generate symbol information file (.st) for API trace, if available. Ignored when not performing an API trace.

Performance Counter mode options (for --perfcounter and --hsapmc):

-1

-c [counterfile] arg	Path to the counter file to enable selected counters (case-sensitive). If not provided, all counters are used. Ignored when performing an API trace. See below for information about the counter file format.
-g [singlepass]	Only allow a single pass when collecting performance counters. Any counters that cannot fit into a single pass will be ignored. If specified, the GPUTime will not be collected, as a separate pass is required to query the GPUTime (OpenCL [™] or DirectCompute only, this is the default for HSA).
-G [nogputime]	Skip collection of GPUTime when profiling a kernel (GPUTime requires a separate pass) (OpenCL [™] or DirectCompute only, this is the default for HSA).
-k [kerneloutput] arg	Output the specified kernel file (OpenCL [™] or DirectCompute only). Valid argument values are: il: output kernel IL files isa: output kernel ISA files cl: output kernel CL files hsail: output kernel HSAIL files asm: output DirectCompute shader ASM files all: output all files
-K [kernellistfile] arg	Path to the kernel list file which contains a case-sensitive list of kernels to profile. If not provided, all kernels will be profiled. See below for information about the kernel list file format.

-s [Character used to separate fields in the
outputseparator	OutputFile. Ignored when performing
] arg	an API trace.
-x [maxkernels] arg (=100000)	Maximum number of kernels to profile.

Trace Summary mode options (for --tracesummary):

-a [atpfile] arg	Path to the .atp file from which to generate summary pages. Optional when performing an API trace. Required if -T is specified when not performing an API trace. The handle of the kernel.
-R [apirulesfile] arg	Path to OpenCL [™] API analyzer configuration file. If not specified, all rules are enabled. Ignored when -tracesummary is not specified. See below for information

Examples

An example to collect OpenCL[™] or DirectCompute performance counters:

CodeXLGpuProfiler -o "/path/to/output.csv" -p -w "/path/to/app/working/directory" "/path/to/app.exe" --device gpu

· An example to collect an OpenCL[™] API trace:

CodeXLGpuProfiler -o "/path/to/output.atp" -t -w "/path/to/app/working/directory" "/path/to/app.exe" --device gpu

• An example to collect HSA performance counters: CodeXLGpuProfiler -o "/path/to/output.csv" -C -w "/path/to/app/working/directory"

"/path/to/app.exe"

• An example to collect an HSA API trace:

CodeXLGpuProfiler -o "/path/to/output.atp" -A -w "/path/to/app/working/directory" "/path/to/app.exe"

• An example to collect an OpenCL[™] API trace with summary pages:

CodeXLGpuProfiler -o "/path/to/output.atp" -t -T -w "/path/to/app/working/directory" "/path/to/app.exe" --device gpu

- An example to generate summary pages from an .atp file: CodeXLGpuProfiler -a "/path/to/output.atp" -T
- An example to generate an occupancy display page: CodeXLGpuProfiler -P "/path/to/occupancy/params/file.txt" -o "path/to/output.html"

After you have used the command line to profile an application., you can view the results within CodeXL using the **Import Session** command in the **CodeXL Explorer**.

The format of the configuration files passed to the --counterfile, --apirulesfile, --apifilterfile, --envvarfile and --occupancydisplay

options can be found in the **Description of Configuration Files** topic.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > GPU Profiler > Using the GPU Profiler > AMDTActivityLogger Library

AMDTActivityLogger Library

The AMDT Activity Logger (previously named CLPerfMarkerAMD) library provides a simple host-code instrumentation API that can help you analyze your OpenCL applications.

It lets you instrument your code with calls to amdtBeginMarker() and amdtEndMarker (). These calls are then used by the GPU Profiler to annotate the host-code timeline in a hierarchical way.

The library also lets you instrument your code with calls to amdtStopProfiling() and amdtResumeProfiling() to control which parts of your application are profiled.

The following screenshot shows an application that has been instrumented with this API. In the image, the rows labeled **Smoke** and **TeapotOGL** under the **Host Thread 4864** branch represent performance markers added to the host code.

Mar-13-2014_17-12-22 (Application Timeline Trace)								
Application Timeline Trace								
Milliseconds	4049. 4047.215 4048.736	118 4050.258	4051.780	4053.302 405	4.823 4056.345	4057.867	4059.388	4060.910
⊟Host								
⊟ Host Thread 4864								
OpenCL	dFinish	qui dFinish		queueAcquir	eGLObj	dFinish		
⊡ Smoke	AMDTTeapotOC	.SmokeSystem::draw			AMDTTeapotOCLSm	okeSystem::draw		
Smoke	AMDTTeapotOCLSr	nokeSystem::compute	nc		AMDTTeapotOCLSmo	keSystem::compute		
⊡ Smoke								
Smoke								
⊡ TeapotOGL				a				
TeapotOGL								
⊡ OpenCL								
□ Context 0 (0x08F70B70)								
□ Queue 0 - Devastator (0x091AED58)								
Data Transfer		Y_BUFFER						
Kernel Execution	FielceEctFieldSerrensit			pplyBu	ioyancy calculateCurlU	applyVorticity	vectFieldVeloc	yBounda
	•							Þ

For more information on this API, see the **AMDTActivityLogger.pdf** file in the **AMDTActivityLogger/Doc** subdirectory under the CodeXL installation directory.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Static Analyzer

Static Analyzer

In Analyze mode, you can compile shaders and kernels for a variety of AMD GPUs and APUs, independent from the GPU/APU that is physically installed on your system, and generate AMD ISA, intermediate language and performance statistics for each target platform. CodeXL Analyzer supports the following inputs:

- OpenCL kernels
- DirectX shaders
- OpenGL and Vulkan programs
- <u>Switching to Analyze mode</u>
- Creating a new project for Analysis
- Working with the new CodeXL Analyzer Explorer Tree
- <u>Working with Programs</u>
- Working with Folders
- <u>Build Options- Defining OpenCL and DirectX build</u> options
- <u>Output Tab</u>
- <u>Kernel Statistics Tab</u>
- <u>Shader Statistics Tab</u>
- <u>Viewing compilation output: IL and ISA</u>
- <u>Export binaries</u>
- <u>Remove items from Project</u>
- <u>Static Analyze Toolbar for OpenCL source files</u>
- <u>Static Analyze Toolbar for DirectX source files</u>
- CodeXLAnalyzer Command Line Interface



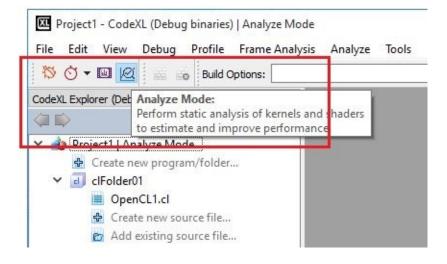


Help > Using CodeXL > Static Analyzer > Switching to Analyze mode

Switching to Analyze mode

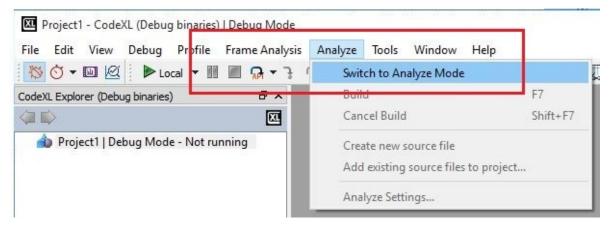
Option 1- Analyze mode button:

Click on the Analyze Mode button in the CodeXL Mode toolbar:



Option 2- Main menu:

Open the Analyze menu from menu bar and select the 'Switch to Analyze Mode' command:



After you switch to Analyze mode, you can also create a new project, open a previously saved project, or load the Teapot or Matrix Multiply samples.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Static Analyzer > Creating a new project for Analysis

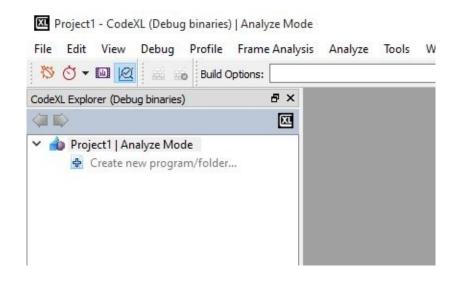
Creating a new project for Analysis

Click on the "File->Create Project", or use the Ctrl+N shortcut. The following CodeXL Project Settings dialog will appear:

Create a new CodeXL Project		×
Create a new CodeXL Project General Debug V Profile > CPU Profile Application Timeline Trace GPU Profile: Perf. Counters Power Profile Frame Analysis	Remote Host Address: Target Application	Project2 Local Host O Remote Host Port: 27015 Test Connection
 Analyze Kernel/Shader Build Options HLSL Build Options 	Application Type: Executable Path: Working Directory: Command Line Arguments: Environment Variables: Source Files Source Files Directories: Source Files Resolution Root Directory:	Desktop Application Windows Store App
Restore Default Settings		OK Cancel

Rename the project, and click on the OK.

After the new project has been created, the CodeXL Analyzer Explorer Tree should appear in the left pane:



© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Static Analyzer > Working with the new CodeXL Analyzer Explorer Tree

Working with the new CodeXL Analyzer Explorer Tree

If you are familiar with the former versions of the Analyzer, you probably noticed that the tree has a different structure than the one used in previous versions. Let's examine the structure of the new CodeXL Analyzer Explorer:

1. **Programs and Folders:** before describing how to technically create Programs and Folders, let's first discuss what those objects are, and why they can be useful.

a. Programs (OpenGL, Vulkan):

As of version 2.0, CodeXL can compile and link together multiple source files for OpenGL and Vulkan. This is especially important when different shaders have mutual impact on one another's ISA and performance statistics. To provide that type of support, CodeXL Analyzer introduced the concept of a Program. There are two types of Programs in CodeXL 2.0:

- Rendering Programs
- Compute Programs

A <u>Rendering Program</u> represents a graphics pipeline, and can have a single shader attached to each of its stages:

- Vertex
- Tessellation Control
- Tessellation Evaluation
- Geometry
- Fragment

A <u>Compute Program</u> represents a compute pipeline, and can have a single compute shader attached to its single stage.

When you build a program that has multiple shaders attached to it, all shaders are being compiled and linked together. This way, you get more

accurate ISA and performance statistics than those generated using previous versions of CodeXL.

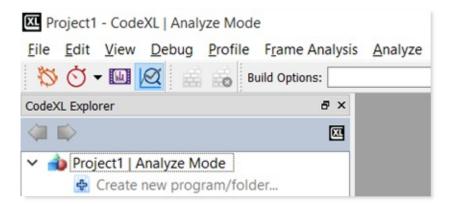
b. Folders (OpenCL, DirectX):

Folders are logical containers of source files. When you build a folder that has multiple source files attached to it, the source files are simply being built one after the other. Unlike programs, there is no kind of interdependency between the source files in a given folder: when a folder is being built, each source file is being compiled independently. Folders can be used to organize the project, by serving as a logical separator. They can also be used to ease the process of comparing build results, since now the build results are being maintained per-folder: you can create two different Folders, each containing the same source files, but have a different configuration (for example, create two DirectX Folders, each with a different shader model). After building the two Folders, you can toggle between the performance statistics of the two Folders to see the differences.

You may ask yourself why CodeXL does not support the concept of DirectX Programs, just like it does for OpenGL and Vulkan. This is a good point. Supporting DirectX Programs is at a high priority in the Analyzer's roadmap, and we will do our best to add that feature in the upcoming versions of the product.

Creating a new Program or Folder

To create a new Program or a Folder, double-click on the "Create new program/folder" item in CodeXL Analyzer Explorer Tree:

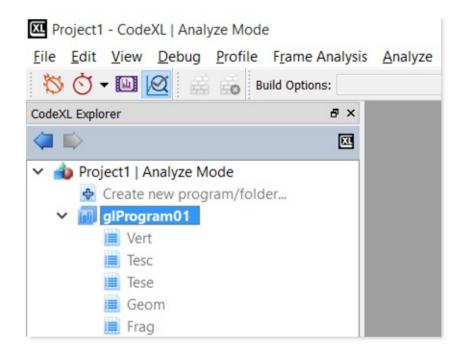


Then, the following dialog would pop-up:

Create a New Program/Folder	×
Select Program/Folder type:	
Vulkan	
Rendering Program	
Compute Program	
OpenGL	
Rendering Program	
Compute Program	
DirectX	
O DirectX Folder	
OpenCL	
OpenCL Folder	
Description:	
An OpenGL Rendering Program is a graphics pipeline that can have a sin shader attached to each of its stages: Vertex, Tessellation Control, Tess Evaluation, Geometry, Fragment. When built, all of the program's shader being compiled and linked together.	ellation
ОК Са	ancel

Select the Program/Folder type of choice, and click OK.

Then, the empty Program/Folder would appear in the Explorer Tree. For Example, if you choose an OpenGL Rendering Program, you will see an empty OpenGL Rendering Program created:



 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.

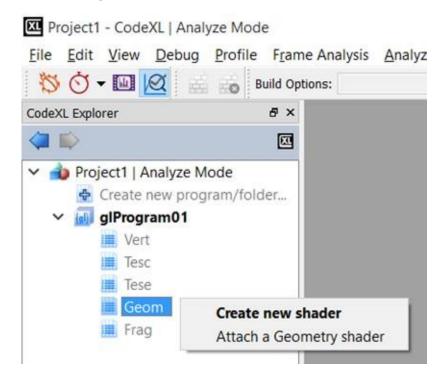




Help > Using CodeXL > Static Analyzer > Working with Programs

Working with Programs

After creating a new program, you will see that it contains an empty placeholder for every pipeline stage. Right-click on any stage to add an existing shader or create a new one:



<u>Note</u>: You can also double-click on a stage to create a new shader and automatically attach it to that Program's stage.

To build the program, right-click on it and select the Build option, or use the F7 shortcut:

Project1 - CodeXL Anal File Edit View Debug	yze Mode Profile Frame Analysis Analyze Te	ools Window
🖏 🔿 • 🖬 🙋 📓	Build Options:	
CodeXL Explorer	ē ×	
4		
 Project1 Analyze M Create new prog 		
V glProgram01	Build glProgram01	F7
Vertex1.ve Tesc Tese	Rename program Remove gIProgram01 from Project	F2 Del
E Geometry	Export Binaries	_
> 🔂 Output (32-b > 👩 glProgram02	pit)	

You can also select the Program and manually click on the Build button in the Analyzer toolbar:

🗷 Pro	oject1	- Code	XL Analy	ze Mode	2	
File	Edit	View	Debug	Profile	Frame Analysis	Analyze
13	Ø.	•	ଷ 🚔	B	uild Options:	

 $\ensuremath{\textcircled{\sc c}}$ 2015 Advanced Micro Devices. All rights reserved.

÷.

i

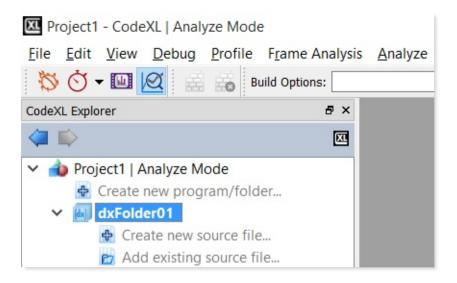




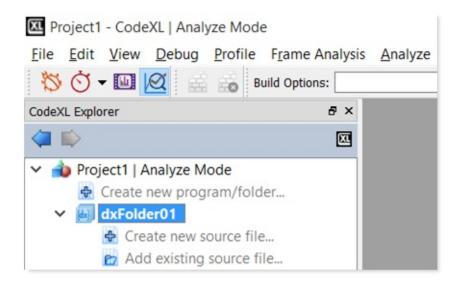
Help > Using CodeXL > Static Analyzer > Working with Folders

Working with Folders

After creating a new OpenCL or DirectX Folder, an empty Folder would be listed in the Explorer Tree:



To create a new source file, and automatically add it to the Folder, double-click on the "Create new source file item..." item of the folder:

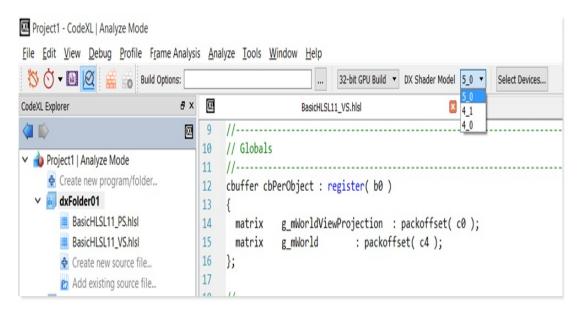


To add an existing source file, and automatically add it to the Folder, double-click on the "Add existing source file item..." item of the folder:

🛛 Project1 - CodeXL Analyze M	de
<u>File Edit View D</u> ebug <u>P</u> rofi	e F <u>r</u> ame Analysis
🏷 🔆 - 💷 🖾 📾	Build Options:
CodeXL Explorer	₽×
4	X
🗸 👈 Project1 Analyze Mode	
🚓 Create new program/f	lder
✓ ຟ dxFolder01	
🖶 Create new source	ile
👩 Add existing source	file

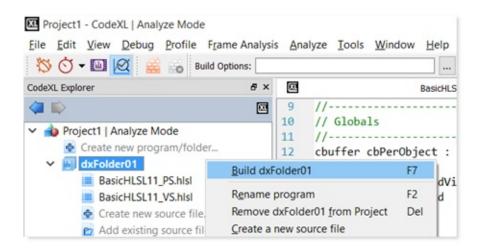
To configure the build properties of a source file under a specific Folder, click on that source file and use the Analyzer toolbar's Type and Entry point drop-down lists. The first sets the type of the shader and the latter specifies the specific target shader (among the shaders in the source file). This configuration is <u>Folder-specific</u>. That is, the same source file can be set with different properties under different Folders. CodeXL will remember those configurations for you.

To configure the build properties of the Folder, click on the Folder and adjust the enabled items in the Analyzer toolbar. For CodeXL 2.0, this is only relevant to the DX Shader Model property of DX Folders:

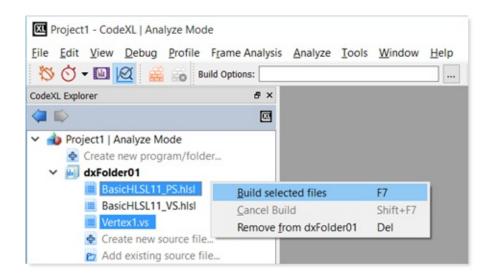


Once set, the DX Shader Model value will hold for all the shaders in the selected Folder. For example, if you choose 5_0 as the DX Shader Model, any D3D vertex shader in that Folder will be compiled using shader model vs 5_0 .

To build the whole Folder, right-click on it and select the Build item:



Unlike the case with Programs, Folders are more flexible as they allow you to build selected source files, without being required to build the whole Folder. To build selected source files, click on the selected source files under the program, while holding the Ctrl key. Then, right-click on one of the selected files and select the build option:



© 2015 Advanced Micro Devices. All rights reserved.

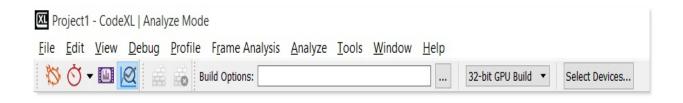




Help > Using CodeXL > Static Analyzer > Selecting target devices

Selecting target devices

CodeXL Analyzer can target a variety of devices, independent of the device that is physically installed on your system. To select the target devices, for which the build would be performed, first click on the Select Devices button in the Analyzer toolbar:



Then, the CodeXL Options dialog would pop-pup with its Analyze tab activated. The devices are grouped by generations. You can use the check boxes to select and remove devices:

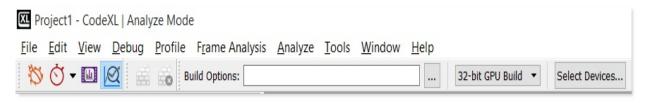
CodeXL Options	×
General GPU Debug CPU Profile Analyze	
Select devices to include them as build targets The displayed devices are devices supported by the installed driver	
Devices	
 > ✓ Carrizo > ☐ Fiji 	
> Iceland > Tonga	
Restore Default Settings OK	Cancel

© 2015 Advanced Micro Devices. All rights reserved.



Build Options- Defining OpenCL and DirectX build options

In the Static Analyze toolbar, you can define specific OpenCL or HLSL build options:



The Build Options box is a place to set compiler build flags such as $-x \operatorname{clc}^{++}$ or -o3. Any compiler build flag can be placed in this box.

You can set the build options by typing the options directly in the designated text box or by using the OpenCL/HLSL Build Options dialog.

OpenCL Build Options Dialog

This dialog will help you choose the correct OpenCL build options for you and hopefully will prevent making spelling mistakes while typing the options manually.

To open the dialog, press The button. The dialog will be opened. You can switch between the "General & Optimization" tab and the "Other" tab to view all the available options. Once you choose an option, the option text is displayed in the "OpenCL Build Command Line" text box that appears below. This string will also appear in the menu bar after you click the OK button. While typing a command in the "OpenCL Build Command Line" text box, you will notice that the relevant controls are being updated accordingly (for example, if you will type "-w", you will be able to see that the "Disable all warnings" check box becomes checked).

General	General		
Debug Profile	Predefined macros		
> CPU Profile Application Timeline Trace	Additional include directories OpenCL format	Default	
GPU Profile: Perf. Counters Power Profile	Disable all warnings Treat any warning as an error	-w -Werror	
Frame Analysis Analyze	Optimization		
 Kernel/Shader Build Options 	Optimization level	Default	
General & Optimization Other	Treat double float-point constant as single one	-cl-single-precision-constant	
HLSL Build Options	Flush denormalized floating point numbers as zeros	-cl-denorms-are-zero	
	Compiler assumes the strict aliasing rules	-cl-strict-aliasing	
	Enable MAD	-cl-mad-enable	
	Ignore the signedness of zero	-cl-no-signed-zeros	
	Allow unsafe optimization	-cl-unsafe-math-optimizations	
	Assume no NaN nor infinite	-cl-finite-math-only	
	Do aggressive Math Optimization	-cl-fast-relaxed-math	
	Correctly round single-precision FP divide sqrt	-cl-fp32-correctly-rounded-divide-sqrt	
	OpenCL Build Command Line		

Usage Example: build options

For building the **tpAdvectFieldScalar.cl** kernel from CodeXL's

AMDTTeaPot sample project, enter the following options:

-D GRID_NUM_CELLS_X=64 -D GRID_NUM_CELLS_Y=64 -D GRID_NUM_CELLS_Z=64 -D GRID_INV_SPACING=1.000000f -D GRID_SPACING=1.000000f -D GRID_SHIFT_X=6 -D GRID_SHIFT_Y=6 -D GRID_SHIFT_Z=6 -D GRID_STRIDE_Y=64 -D GRID_STRIDE_SHIFT_Y=6 -D GRID_STRIDE_Z=4096 -D GRID_STRIDE_SHIFT_Z=12 -I path_to_example_src

On windows, *path_to_example_src* should be:

C:\Program Files\CodeXL\Examples\Teapot\res

On Linux, *path_to_example_src* should be:

/opt//CodeXL/bin/examples/Teapot/AMDTTeaPotLib/AMDTTeaPotLib/

Adding the option '-h' will dump the list of OpenCL compiler available options in the output tab. For additional details, 'Compile Build Options' Appendix.

Build Options

General Options			
-D	Predefined macros	Predefine macros should be sepa the Predefined macro needs to in enclose the macro within parent	
-I	Additional include directories.	Additional include directories sh separated by ';'. If the directory space, enclose the path within p	
-x clc,-x clc++	OpenCL format		
-w	Disable all warnings	Inhibit all warning messages.	
-Werror	Treat any warning as an error	Make all warnings into errors.	
Optimization Options			
-00,-01,-02,-	Optimization		

03,-04,-05	level	
-cl-single- precision- constant	Treat double float-point constant as single one	Treat double precision floating as single precision constant
-cl-denorms- are-zero	Flush denormalized floating point numbers as zeros	This option controls how singl double precision denormalized handled. If specified as a build single precision denormalized flushed to zero and if the option double precision is supported, denormalized numbers may al zero. This is intended to be a p and the OpenCL compiler can flush denorms to zero if the de single precision (or double pre- denormalized numbers. This o for single precision numbers is not support single precision de numbers i.e. if CL_FP_DENOR in CL_DEVICE_SINGLE_FP_CO option is ignored for double pre- if the device does not support or if it does support double pre- cL_FP_DENORM bit is not set CL_DEVICE_DOUBLE_FP_CO only applies to scalar and vect precision floating-point variab computations on these floating inside a program. It does not a from or writing to image object
-cl-strict- aliasing	Compiler assumes the strict	This option allows the compile strictest aliasing rules.

	aliasing rules	
-cl-mad- enable	Enable MAD	Allow a * b + c to be replaced by mad computes a * b + c with rec For example, some OpenCL devi mad as truncate the result of a * adding it to c.
-cl-no-signed- zeros	Ignore the signedness of zero	Allow optimizations for floating- arithmetic that ignore the signed IEEE 754 arithmetic specifies th distinct +0.0 and -0.0 values, wh prohibits the simplification of ex as $x+0.0$ or $0.0*x$ (even with -cl- only). This option implies that th zero result isn't significant.
-cl-unsafe- math- optimizations	Allow unsafe optimization	Allow optimizations for floating- arithmetic that (a) assume that a results are valid, (b) may violate standard and (c) may violate the numerical compliance requireme in section 7.4 for single-precision point, section 9.3.9 for double-pu floating-point, and edge case bel section 7.5. This option includes signed-zeros and -cl-mad-enable
-cl-finite- math-only	Assume no NaN nor infinite	Allow optimizations for floating-j arithmetic that assume that argu results are not NaNs or ±?. This violate the OpenCL numerical cc requirements defined in in section single-precision floating-point, so double-precision floating-point, a behavior in section 7.5.
		Sets the optimization options -cl

-cl-fast- relaxed-math	Do aggressive Math Optimization	only and -cl-unsafe-math-optimiz allows optimizations for floating arithmetic that may violate the I standard and the OpenCL nume compliance requirements define specification in section 7.4 for si floating-point, section 9.3.9 for c precision floating-point, and edg behavior in section 7.5. This opt preprocessor macro _FAST_RELAXED_MATH to b the OpenCL program.
-cl-fp32- correctly- rounded- divide-sqrt	Correctly round single- precision FP divide & sqrt	The -cl-fp32-correctly-rounded-d option to clBuildProgram or clCc allows an application to specify † precision floating-point divide (x sqrt used in the program source rounded. If this build option is n the minimum numerical accurac precision floating-point divide ar defined in section 7.4 of the Ope specification.\nThis build option specified if the CL_FP_CORRECTLY_ROUNDED is set in CL_DEVICE_SINGLE_F1 defined in in the table of allowec param_name for clGetDeviceInfc that the program is being build. clBuildProgram or clCompilePro to compile the program for a dev fp32-correctly-rounded-divide-sc specified and CL_FP_CORRECTLY_ROUNDED is not set for the device.
Other Optio	DNS	0

-cl-std	CL version supported	Determine the OpenCL C langua use. A value for this option must Valid values are:\nCL1.1 - Suppo C programs that use the OpenCl features defined in section 6 of t specification.\nCL1.2 - Support a programs that use the OpenCL C features defined in section 6 of t specification.
-cl-kernel- arg-info	Kernel argument info	This option allows the compiler to information about the argument in the program executable. The information stored includes the name, its type, the address and qualifiers used. Refer to the dese clGetKernelArgInfo for information.
-create- library	Create library	Create a library of compiled bina in input_programs argument to a
-enable-link- options	Enable link options	Allows the linker to modify the li based on one or more link option in Program Linking Options, bel- library is linked with a program This option must be specified wi library option.
-g	Produce debugging information	This is an experimental feature t use the GNU project debugger, (kernels on x86 CPUs running Lii cygwin/minGW under Windows. details, see Chapter 3, "Debuggi This option does not affect the d optimization of the OpenCL code
	Specify that	

-fper-pointer- uav -fno-per- pointer-uav	UAV per pointer should be used (HD5XXX and HD6XXX series GPU's only)	
-fbin-bif30 -fno-bin-bif30	Allow OpenCL binary to be BIF3.0 format	
-fbin-encrypt -fno-bin- encrypt	Generate an encrypted OpenCL binary (not by default)	
-save-temps	Store temporary files in current directory	This option dumps intermediate files, such as IL and ISA code, fo kernel. If <prefix> is not given, are saved in the default tempora (the current directory for Linux, <user>\\AppData\\Local for Win <prefix\\> is given, those tempo saved with the given <prefix>. I an absolute path prefix, such as C:\\your\\work\\dir\\mydumppref temporaries are saved under C:\\your\\work\\dir, with mydum] prefix to all temporary names. Fi under the default directory</prefix></prefix\\></user></prefix>
-fuse-jit	Use JIT for CPU target	

-fno-use-jit	(disable if debugging is enabled	
-fforce-jit -fno-force-jit	Force use JIT for CPU target (even if debugging is enabled)	
-fdisable-avx -fno-disable- avx	Disable AVX code generation	
-ffma-enable -fno-fma- enable	Enable fma for a*b+c	
-fuse-native	Replace math function calls with native version	

HLSL Build Options Dialog

This dialog will help you choose the correct HLSL build options for you and hopefully will prevent making spelling mistakes while typing the options manually.

To open the dialog, press The Button. The dialog will be opened. Click the "HLSL Build Options" node to view the available options.

Once you choose an option, the option text is displayed in the "HLSL Build Command Line" text box that appears below.

This build option string will also appear in the toolbar's build options box after you click the OK button.

As an alternative to selecting options through the radio buttons, it is possible to type a command in the "HLSL Build Command Line" text box. Build options types in the text box will automatically be translated to update of the relevant controls accordingly. For example, typing "D3DCOMPILE_DEBUG" in the lower text box automatically updates the "Debug" check box to be checked.

General	General			^
Debug Profile CPU Profile Application Timeline Trace GPU Profile: Perf. Counters Power Profile	D3D Compiler DLL	D3DCompiler_47.dll (bundled with CodeXL)		•
	 FXC Tool Predefined macros Additional include directories 	C:\Program Files (x86)\Windows Kits\8.1\bin\x64\fxc	.exe	×
Frame Analysis	HLSL Build Options	D3D Compiler DLL	FXC Tool	
Analyze	Avoid Flow Control	D3DCOMPILE_AVOID_FLOW_CONTROL	/Gfa	- 1
✓ Kernel/Shader Build Options	Debug	D3DCOMPILE_DEBUG	/Zi	
General & Optimization	Enable Backwards Compatibility	D3DCOMPILE_ENABLE_BACKWARDS_COMPATIBILITY	/Gec	
Other HLSL Build Options	Enable Strictness	D3DCOMPILE_ENABLE_STRICTNESS	/Ges	
THESE BUILD OPTIONS	Force Pixel Shader Optimization Off	D3DCOMPILE_FORCE_PS_SOFTWARE_NO_OPT		
	Force Vertex Shader Optimizations Off	D3DCOMPILE_FORCE_VS_SOFTWARE_NO_OPT		
	IEEE Strictness	D3DCOMPILE_IEEE_STRICTNESS	/Gis	
	No Preshader	D3DCOMPILE_NO_PRESHADER	/Op	
	Optimization Level	1 (Default)		٠
	Pack Matrix Column Major	D3DCOMPILE_PACK_MATRIX_COLUMN_MAJOR	/Zpc	
	Pack Matrix Row Major	D3DCOMPILE_PACK_MATRIX_ROW_MAJOR	/Zpr	
	Dartial Provision	NONCOMONE ENDTINI DECORTONI	Iran	
	HLSL Build Command Line - 0X0			

Build Options

The compilation of DirectX shaders can be executed either by directly referencing the D3D compiler DLL or by going through Microsoft's FXC tool.

The CodeXL installation includes a copy of the Microsoft DirectX compiler DLL: d3dcompiler_47.dll. You may specify a different path if you want CodeXL to use a different d3dcompiler module. If you select the FXC compiler tool, you must specify a path to the location of FXC.exe.

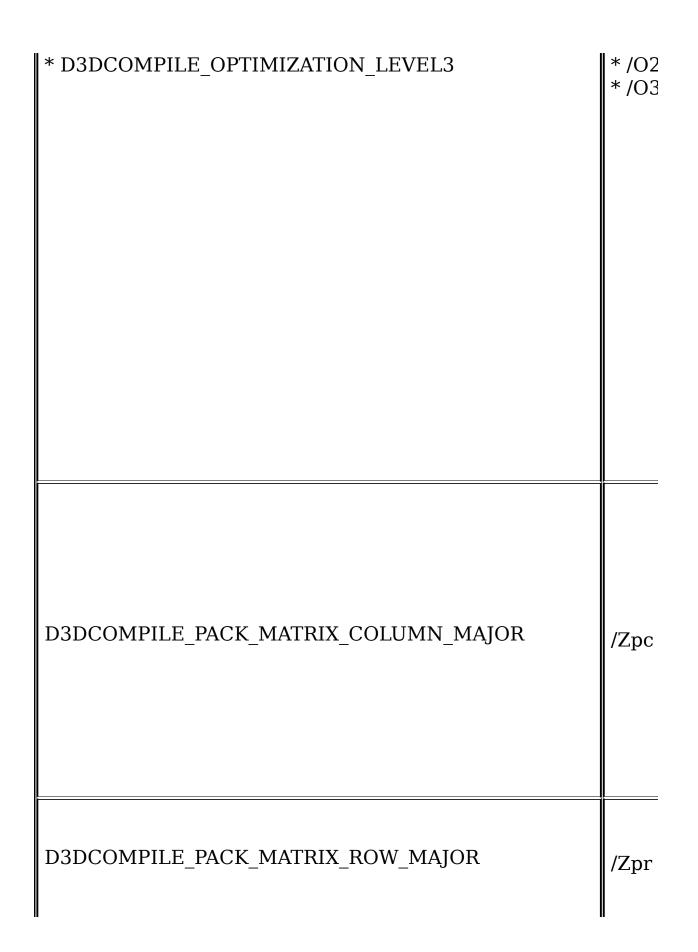
To select the path of the compiler module, click the 'Browse..." option from the combo-box. When selecting Browse, a dialog box will open for selecting the compiler file.

- For D3D compiler any file called d3compiler_*.dll can be selected.
- For FXC compiler only files named FXC.exe can be selected.

D3D compile command	FXC
	com
-D	
-I	

D3DCOMPILE_AVOID_FLOW_CONTROL	/Gfa
D3DCOMPILE_DEBUG	/Zi
D3DCOMPILE_ENABLE_BACKWARDS_COMPATIBILITY	/Gec
D3DCOMPILE_ENABLE_STRICTNESS	/Ges
D3DCOMPILE_FORCE_PS_SOFTWARE_NO_OPT	
D3DCOMPILE_FORCE_VS_SOFTWARE_NO_OPT	

D3DCOMPILE_IEEE_STRICTNESS	/Gis
D3DCOMPILE_NO_PRESHADER	/Op
* D3DCOMPILE_SKIP_OPTIMIZATION * D3DCOMPILE_OPTIMIZATION_LEVEL0 * (no flag for default optimization) * D3DCOMPILE_OPTIMIZATION_LEVEL2	* /Od * /O0 * (I for d * /O1



D3DCOMPILE_PARTIAL_PRECISION	/Gpp
D3DCOMPILE_PREFER_FLOW_CONTROL	/Gfp
D3DCOMPILE_RESOURCES_MAY_ALIAS	/res_
D3DCOMPILE_SKIP_VALIDATION	/Vd

D3DCOMPILE_WARNINGS_ARE_ERRORS	/WX
	/Lx
	/Ni
	/No



 \cdot Note: some of the flags are only relevant to the FXC tool.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Static Analyzer > Output Tab

Output Tab

The compiler output appears in the Output tab. The example below shows successful builds (no warnings or errors) for 4 devices.

Output	6 x
======================================	٨
======================================	
1> DCT_Kernels for Bonaire:	
Building for Bonaire succeeded.	
2> DCT_Kernels for Capeverde:	
Building for Capeverde succeeded.	
3> DCT_Kernels for Carrizo:	
Building for Carrizo succeeded.	
======================================	
======================================	
1> BitonicSort_Kernels for Bonaire:	
Building for Bonaire succeeded.	
2> BitonicSort_Kernels for Capeverde:	
Building for Capeverde succeeded.	
3> BitonicSort_Kernels for Carrizo:	
Building for Carrizo succeeded.	
======================================	

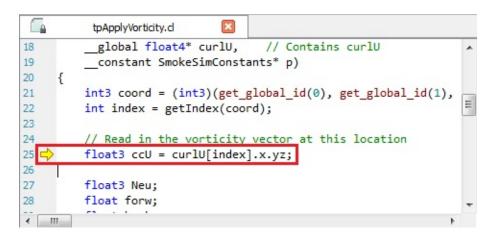
If there were errors, the output will display the error and the line where the error occurred:

 Output
 Image: Computer Park - ------

 Compiling for WinterPark - ------ Failed!

 OpenCL Compile Error: clBuildProgram failed (CL_BUILD_PROGRAM_FAILURE).
 Image: Clause Compiler Compile

Double clicking on an error navigates the user to the Source Code view, displaying the kernel source code:



© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Static Analyzer > Kernel Statistics Tab

Kernel Statistics Tab

The statistics tab gives detailed statistics for the selected kernel for each target device.

To open the statistics tab, expand the desired kernel in the project tree, and double-click the statistics node:

ocal Workgroup: X: 4 Y: 4	Z: 4			Dynamic LDS	Usage: 0	bytes				i-iuhidi	1000
lesource				Usage				(Constraint on N	/lax Waves per SIMI	D (1-10)
GPRs (0-102)		24 Registers 10									
(GPRs (0-256) DS size (0-32,768)		11 Registers 10 2,048 bytes (Static: 2,048 bytes, Dynamic: 0 bytes) 8									
Amount of LDS available with	out reducing c			stic: 2,048 byte	s, Dynamic: 0 i	oytes)			8		
-				tive concurre	ncy constrain	t (Max waves	per SIMD):		8		
lesource	Reco	ommended Us	age		Usage		P	erformance In	npact		
5A Size		<=32KB			608 byte	5			mmended usa	ige	
cratch Registers		0			0		•	Meets reco	mmended usa	ige	
The offerst of second second second second	number of cond	current waves									
The effect of resource usage on the Max waves/SIMD: Num of SGPRs used:	10 <=48	9 49-56	8 57-64	7 65-72	6 73-80	5 81-96	4 96<				
Max waves/SIMD: Num of SGPRs used:	10							3	2	1	
Max waves/SIMD: Num of SGPRs used: Max waves/SIMD:	10 <=48	49-56	57-64	65-72	73-80	81-96	96<	3 65-84	2 85-128	1 128<	
Max waves/SIMD: Num of SGPRs used: Max waves/SIMD: Num of VGPRs used:	10 <=48 10	49-56 9	57-64 8	65-72 7	73-80 6	81-96 5	96< 4			1.	
Max waves/SIMD:	10 <=48 10 <=24 10 <=1,638	49-56 9 25-28 9 1,639-1,820	57-64 8 29-32 8 1,821-2,048	65-72 7 33-36 7 2.049-2.340	73-80 6 37-40	81-96 5 41-48	96< 4 49-64	65-84	85-128	128<	

You can see in this statistics output that Northern Islands devices do not use scalar registers (SGPRs and MaxSGPRs are N/A). Northern Islands VGPRs are quad-sized so the 20 VGPRs actually represent 80 float values. You can also see that Southern Islands devices use 44 scalar GPRs that are shared across the wavefront and 49 VGPRs that are used per thread equaling 49 float values.

This view puts the emphasis on giving the programmer the wave constraints based on the SGPRs, VGPRs and LDS size.

In the upper section there is a table that shows the current constraints based on the kernels information for the current selected device.

In the lower section, there is a reference table to help the programmer see the effect of the resources usage on the number of concurrent waves.

The LDS is constructed from the static & dynamic values. The dynamic value can be defined by the user to and its possible effect on the constraint can be immediately viewed. LDS is also affected by local workgroups size, so the values can be changed in order to see the impact on the performance of the kernel.

The middle table shows if the "ISA size" and "Scratch Registers" might have impact on the performance and what are the recommended values.

 $[\]ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Static Analyzer > Shader Statistics Tab

Shader Statistics Tab

The statistics tab gives detailed statistics about the selected shader for each target device.

To open the statistics tab expand the desired shader in the project tree, and double-click the statistics node:

There is no support for shader statistics for v5 and earlier generation devices.

Statistics page for GCN devices (v6 and later generations):

lesource		Z:									
			Usage				Constraint on Max Waves per SIMD (1-10)				
GPRs (0-102)						4 Registe	ers				10
/GPRs (0-256)						3 Registe	ers				10
				Ef	fective c	oncurren	cv constr	aint (Ma	waves p	er SIMD):	10
SA Size Scratch Registers			<=32KB 0				20	bytes 0			Meets recommended usage Meets recommended usage
Performance Reference Tabl	es										
The offect of recourse upper	on the p	mhar of a	and grant								
The effect of resource usage Max waves/SIMD:				<u></u>	6	5	1	1			
The effect of resource usage Max waves/SIMD: Num of SGPRs used:	e on the nu 10 <=46	9 47-54	8 55-62	waves 7 63-70	6 71-78	5 79-94	4 94<				
Max waves/SIMD:	10	9	8	7				3	2	1	

The displayed information is explained in the following table:

Column Name	Explanation
SGPRs	The number of scalar General Purpose Registers allocated by the shader
VGPRs	The number of vector General Purpose Registers allocated by the kernel. ReqdWorkGroupX - Required workgroup X size specified for kernel. N/A without optional attribute((reqd_work_group_size(X, Y, Z)))
ISA size	Compiled code size

Bogistors	The number of scratch registers used by the kernel. If this value is bigger than 0, the shader may be incurring a performance penalty
MaxSGPRs	The maximum number of scalar General Purpose Registers per kernel supported by the device
MaxVGPRs	The maximum number of vector General Purpose Registers per kernel supported by the device

This view puts the emphasis on giving the programmer the wave constraints based on the SGPRs and VGPRs.

In the upper section there is a table that shows the current constraints based on the shaders information for the current selected device.

In the lower section there is a reference table to help the programmer see the effect of the resources usage on the number of concurrent waves.

The middle table shows if the "ISA size" and "Scratch Registers" might have impact on the performance and what are the recommended values.

 $[\]ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Static Analyzer > Viewing compilation output: IL and ISA

Viewing compilation output: IL and ISA

The performance statistics tab will be opened automatically when the build process is over. To view the compilation output, double click the node of the desired ASIC in the explorer tree, under the Program/Folder and configuration (32-bit or 64-bit):

S

This will open a tab containing the source code, the IL and the ISA. The program source code and the IL code will be presented as standard text documents. The ISA will be presented in the "Enhanced ISA View" for GCN devices, and as a standard text document for pre-GCN devices.

The context menu also enables showing/hiding line numbers for each source code/IL/ISA tab.

 $[\]ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Static Analyzer > Navigating through ISA code with the Enhanced ISA View

Navigating through ISA code with the Enhanced ISA View

Using this view, you can inspect the ISA code of GCN devices and see the estimation for instruction cost in clock cycle. The view contains 5 columns:

- **Address**: the instruction's offset within the program (in bytes)
- **Opcode**: the operation to be performed
- **Operands**: the data for the operation
- **Cycles**: the number of clock cycles which are required by a Compute Unit in order to process the instruction for a 64-thread Wavefront, while neglecting the system load and any other runtime-related factor.
- **Instruction** Type: the category of instructions to which the instruction belongs
- **Hex**: binary representation of the instruction, in hexadecimal format

Notes:

1. Note that code labels which appear in the Operands column are clickable. By clicking on a label link, you can navigate to the label's spot in the code.

2. Note that this view is only available for GCN devices. For pre-GCN devices, the plain textual ISA view will be displayed.

 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Static Analyzer > Export binaries

Export binaries

You can export the binaries of the last build by right clicking on the folder or file in the explorer tree and selecting the "Export binaries..." option.

When you select the "Export Binaries..." option, an "Export Binaries" dialog will open where you can select the destination folder as well as which parts of the binaries will be included.

You can change the default base name by entering desired name into "Base file name" text field. All binary files will be created with the chosen base name followed by device name.

All the devices that were built will be exported.

* Notice: this option in only supported in OpenCL mode.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Static Analyzer > Remove items from Project

Remove items from Project

Programs, Folders and source files can be removed from the project. To remove an item from the project, right-click on it and select the Remove option.

 $\ensuremath{\textcircled{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Static Analyzer > Static Analyze Toolbar – for OpenCL source files

Static Analyze Toolbar – for OpenCL source files

Build command
Kernel Build options
Opens the "Build Options" dialog
The bitness of the compilation process (for example, choose OpenCL, 64-bit to compile the files in using the 64-bit OpenCL compiler).
Displays the last selected kernel for the selected file. Jumps to the selected kernel in the active source if the source code is open.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Static Analyzer > Static Analyze Toolbar – for DirectX shaders

Static Analyze Toolbar – for DirectX shaders

	Build only command
	Shader Build options
	Opens the "Build Options" dialog
	The bitness of the compilation process (for example, choose
	OpenCL, 64-bit to compile the
	files in using the 64-bit OpenCL
	compiler).
	The selected shader model. This
	parameter is set once at the
	Folder level, and holds for all the
	shaders in that Folder.
	The type of the selected shader
	(Vertex, Pixel, Hull, etc.)
	The target shader for the
Entry point: RenderPS	compilation process.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Static Analyzer > CodeXLAnalyzer Command Line Interface

CodeXLAnalyzer Command Line Interface

CodeXLAnalyzer (formerly CodeXLKernelAnalyzer) enables compiling and generating performance statistics for OpenCL kernels, DirectX Shaders and OpenGL shaders. The compilation and statistics generation processes can be targeted at a variety of AMD GPUs and APUs, regardless to the actual GPU/APU type that is installed on your system. The application capabilities were expanded. OpenCL Kernels as well as DirectX shaders can now be compiled.

CodeXLAnalyzer can be executed from a shell window using the CodeXLAnalyzer console application which is located in the CodeXL installation directory.

<u>Note:</u> On Windows, CodeXLAnalyzer is available in both 32-bit and 64-bit versions. To execute the compilation and statistics generation in 64-bit, invoke CodeXLAnalyzer-x64.exe. On Linux, only a 64-bit version is available.

<u>Using CodeXLAnalyzer Command Line Interface to</u> <u>compile OpenCL Kernels</u>

<u>Using CodeXLAnalyzer Command Line Interface to</u> <u>compile DirectX shaders</u>

<u>Using CodeXLAnalyzer Command Line Interface to</u> <u>compile OpenGL and Vulkan programs</u>

<u>Generating and interpreting CodeXLAnalyzer CLI's live</u> <u>register analysis report</u>

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Static Analyzer > CodeXLAnalyzer Command Line Interface > Using CodeXLAnalyzer Command Line Interface to compile OpenCL Kernels

Using CodeXLAnalyzer Command Line Interface to compile OpenCL Kernels

OpenCL is the default language for CodeXLAnalyzer, so in order to compile OpenCL kernels specifying the input source code language is optional.

CodeXLAnalyzer uses the actual AMD OpenCL Driver installed on the computer, i.e the Catalyst driver to perform offline compilation.

If no GPU is present, the OpenCL driver installed with APP SDK can be used.

Details of available commands:

· · · · · · · · · · · · · · · · · · ·	
-h	View available options
-11	CodeXLAnalyzer.exe -h
-S	Specify the source language for the compilation. "cl" is the default which means that for OpenCL kernel compilation there is no need to specify the -s switch
	CodeXLAnalyzer.exe -s cl -l
-I [list-asics]	List known ASIC targets.
	CodeXLAnalyzer.exe -l
verbose	View supported ASICS with detailed marketing names
verbuse	CodeXLAnalyzer.exe -lverbose
version	Print version string.
version	CodeXLAnalyzer.exeversion
-a [analysis] arg	Path to output analysis file. Requireskernel.

	CodeXLAnalyzer.exe foo.clkernel myKernelanalysis foo.csv
-c [asic] arg	Which ASIC to target. Repeatable.
	CodeXLAnalyzer.exe foo.clkernel myKernel -isa foo.isaasic Bonaire
	List the kernels functions available in the specify of file
list-kernels	List the kernels functions available in the specify cl file CodeXLAnalyzer foo.cllist-kernels
	Path to output ISA disassembly file(s). This command requires compilation switches identifying the required kernel - -kernel
	CodeXLAnalyzer.exe foo.clkernel Fooisa c:\files\Foo
isa arg	Detailed explanation: kernel FooKernel: compile and get ISA for "Foo" kernel isa c:\files\Foo: specify the designated output location and prefix. This will creates files such as c:\files\Foo-Bonaire.amdisa, c:\files\Foo-Hawaii.amdisa, etc.
il arg	Path to output IL file(s). Requireskernel.
	CodeXLAnalyzer foo.clkernel myKernelil foo.ilasic Tahity
debugil arg	Path to output Debug IL file(s). CodeXLAnalyzer foo.clkernel myKerneldebugil foo.debugil
	CodexLAnaryzer 100.cfkerner myKernerdebugn 100.debugn
	Path to output Metadata file(s). Requireskernel.
metadata arg	CodeXLAnalyzer foo.clkernel myKernel metadata foo.metadata
-b [binary]	Path to binary output file(s).
arg	CodeXLAnalyzer foo.clkernel myKernel –b foo.bin
suppress	Section to omit from binary output.
arg	Repeatable.
	CodeXLAnalyzer foo.clkernel myKernel –suppress .source –b foo.bin
	4
-k [kernel] arg	Kernel to analyze or make IL or ISA.
k kenner Jurg	CodeXLAnalyzer foo.clkernel myKernel –b foo.bin
 OpenCLoption	OpenCL compiler options. Repeatable.
	CodeXLAnalyzer foo.clkernel myKernelisa foo.isaOpenCLoption -cl-

arg	enable-madOpenCLoption -w
-D [define] arg	Define symbol or symbol=value. Repeatable.
	CodeXLAnalyzer foo.clkernel myKernelisa foo.isa –D myDefine
	Override to default separator for analysis items.
csv-separator arg	CodeXLAnalyzer foo.clkernel myKernelanalysis foo.csv –csv-
	_ separator #
livereg arg	Path to the live register analysis output file (note that "isa arg" must be used in conjunction with thelivereg switch for live register analysis to be performed, since the live register analysis engine works by analyzing the ISA disassembly). Note: this is a beta feature of CodeXLAnalyzer CLI. You can find more info about it in the "Generating and Interpreting CodeXLAnalyzer CLI's Live Register Analysis Report"
	CodeXLAnalyzer –s cl –c Bonairekernel myKernelisa foo.isalivereg fooLiveRegFile.txtil fooIl.il myClFile.cl

Usage examples:

Create binary files output/foo-ASIC.bin for foo.cl.

CodeXLAnalyzer foo.cl --bin outdir/foo

List the kernels available in foo.cl.

CodeXLAnalyzer foo.cl --list-kernels

Generate ISA and performance statistics for all ASICs for kernel myKernel which is defined in foo.cl

CodeXLAnalyzer foo.cl --kernel myKernel --analysis foo.csv

List the ASICs that the runtime supports.

CodeXLAnalyzer -- list-asics

Generate ISA and IL code for Cypress (ASIC) for kernel myKernel which is defined in source file foo.cl

CodeXLAnalyzer foo.cl --kernel myKernel --il foo --isa foo --asic Cypress

Generate ISA code and live register analysis report for Iceland (ASIC), for kernel myKernel which is defined in source file foo.cl

CodeXLAnalyzer foo.cl --kernel myKernel --livereg livereg.txt --isa foo --asic Iceland

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Static Analyzer > CodeXLAnalyzer Command Line Interface > Using CodeXLAnalyzer Command Line Interface to compile DirectX shaders

Using CodeXLAnalyzer Command Line Interface to compile DirectX shaders

CodeXLAnalyzer command line tool supports offline compilation and statistics generation for DirectX shaders. Naturally, it is supported on Windows only.

The Analyzer works in 2 stages:

- By default, CodeXLAnalyzer compiles the shader using the D3D Compiler. CodeXL ships with a default compiler (d3dcompiler_47.dll). Unless the --DXLocation command line switch is specified, CodeXL will use the default compiler. The compilation can also go through Microsoft's FXC tool instead of directly through the D3D compiler. To use FXC, you need to specify the -FXC command line switch with the location of FXC.exe.
- 2. CodeXLAnalyzer compiles the D3D ASM code generated by the D3D Compiler into AMD ISA, and generates statistics.

Regardless to the chosen compilation chain, all build errors and warnings will be printed in the command line window.

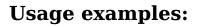
Details of available commands:

Details of available commands:		
-h	View available options	
	CodeXLAnalyzer.exe -h	
-S	Specify the source language for the compilation. For all DX options, you need to specify the source since OpenCL is the default language. Available options: DXAsm, HLSL, CL (default) CodeXLAnalyzer.exe -s hlsl	

-s hlsl –l	View supported ASICS for DirectX (also- this is default ASICs list for compilation) CodeXLAnalyzer.exe -s hlsl –l
-s hlsl -l verbose	View supported ASICS with detailed marketing names CodeXLAnalyzer.exe -s hlsl -l - verbose
isa arg	Path to output ISA disassembly file(s). This command requires compilation switches such as -f, -p.
	CodeXLAnalyzer.exe -s hlsl -f VsMain -p vs_5_0 -p vs_5_0 c:/files/myShader.fxisa c:/files/myShader.isa
	Detailed explanation: -s HLSL: run HSLS compilation (vs CL) -f VsMain: compile and get ISA for "VsMain" function -p vs_5_0: the target in D3DCompile: ps_5_0, vs_5_0, gs_5_0 c:/files/myShader.fx: the file to compile -isa c:/files/myShader.isa: specify the designated output file.
-c [asic] arg	By default, compilation will be done to all default available devices (run -s hlsl -l to view the list). This option enables you to choose which ASIC to target. This option is repeatable.
	CodeXLAnalyzer.exe -s hlsl -f VsMain -p vs_5_0 c:/files/myShader.fxisa c:/files/myShader.isa -c Hawaii -c Kaldini
	0
-a [analysis]	Path to output analysis file. Requires compilation.

	CodeXLAnalyzer.exe -s hlsl -f VsMain c:/files/myShader.fx -p vs_5_0 -a c:/files/myShader.csv
	5 1 5
-D	CodeXLAnalyzer.exe -s hlsl -f VsMain c:/files/myShader.fxisa c:/files/myShader.isa -p vs_5_0 –D MyDefine
	Compile using DXFlags
DXFlags	CodeXLAnalyzer.exe -s hlsl -f VsMain c:/files/myShader.fxisa c:/files/myShader.isa -p vs_5_0DXFlagx 1
version	View the Catalyst driver version installed CodeXLAnalyzer_d.exe -s hlsl –version
	CodexLAnaryzer_d.exe -s nisi =version
DXLocation	Compile with a specific D3Dcompiler DLL. Note that the path need to be in quotes if it contains spaces.
	CodeXLAnalyzer.exe -s hlsl -f VsMain -p vs_5_0 c:/files/myShader.fxisa c:/files/myShader.isa -c HawaiiDXLocation "C:\Program Files (x86)\Windows Kits\8.1\bin\x86\d3dcompiler_47.dll"
-s DXAsm	Compile from a blob. In this case, the application assumes D3DCompilation is done so it skips this and does the AMD DX Compilation only.
	CodeXLAnalyzer.exe -f VsMain -s DXAsm -p vs_5_0 c:/files/myShader.objisa c:\temp\dxTest.isa
·	
	Compile using FXC. You need to provide the full FXC path and arguments, and need to use /Fo switch. Also, the output of the FXC

	file should be the input file for KA. Use with the DXAsm switch.			
FXC	CodeXLAnalyzer.exe -s DXAsm -f VsMain -p vs_5_0 c:/files/myShader.fxisa c:\files\myShader.isa -c tahiti - -FXC "\"C:\Program Files (x86)\Windows Kits\8.1\bin\x86\fxc.exe\" /E VsMain /T vs_5_0 /Fo c:/files/myShader.obj c:/files/myShader.fx"			
-s DXAsmT	Compile from a blob represented as text (DX ASM as Text) . In this case, the application assumes D3DCompilation is done so it skips this and does the AMD DX Compilation only.			
	CodeXLAnalyzer.exe -s DXAsmT -f VsMain -p vs_5_0 c:/files/myShaderAsText.asmisa c:\files\myShader.isa -c tahiti			
	When using this option, the MS Compilation output will be saved in the designated file			
 DumpMSInterm arg	ediate CodeXLAnalyzer.exe -s hlsl -f ps_main c:\temp\Pixel.pshisa c:\temp\dxOutput.isa -c Tahiti -c Bonaire -p ps_5_0 DumpMSIntermediate			
	c:\temp\DumpMSIntermediate.txt			
	Path to the live register analysis output file (note that "isa			
livereg arg	arg" must be used in conjunction with thelivereg switch for live register analysis to be performed, since the live register analysis engine works by analyzing the ISA disassembly). Note: this is a beta feature of CodeXLAnalyzer CLI. You can find more info about it in the "Generating and Interpreting CodeXLAnalyzer CLI's Live Register Analysis Report"			
	CodeXLAnalyzer.exe -s hlsl -c Fiji -f VSMain -p vs_5_0isa c:\temp\.txt - -livereg c:\temp\lreg.txt c:\temp\dx\BasicHLSL11_VS.hlsl			



 Suppose that you would like to compile and generate the ISA code of a DirectX pixel shader (C:\Users\shaders\Render.hlsl), using a the default D3D compiler that ships with CodeXL, and would like the output files which contain the ISA code to be located at c:\temp\ and be named myISA-<device-name>.isa:

CodeXLAnalyzer.exe -s hlsl -p ps_5_0 -f RenderPS --isa c:\temp\myISA.isa C:\Users\shaders\Render.hlsl

2. Suppose that you would like to compile and generate the ISA code of a DirectX pixel shader (C:\Users\shaders\Render.hlsl), using a specific D3D compiler (C:\Program Files (x86)\Windows Kits\8.1\bin\x86\d3dcompiler_47.dll), and would like the output files which contain the ISA code to be located at c:\temp\ and be named myISA-<device-name>.isa:

CodeXLAnalyzer.exe -s hlsl -p ps_5_0 -f RenderPS --DXLocation "C:\Program Files (x86)\Windows Kits\8.1\bin\x86\d3dcompiler_47.dll" --isa c:\temp\myISA.isa C:\Users\shaders\Render.hlsl

3. Suppose that you would like to compile and generate the ISA code of a DirectX pixel shader (C:\Users\shaders\Render.hlsl) only for "Iceland", using a specific D3D compiler (C:\Program Files (x86)\Windows Kits\8.1\bin\x86\d3dcompiler_47.dll), and would like the output files which contain the ISA code to be located at c:\temp\ and be named myISA-<device-name>.isa:

CodeXLAnalyzer.exe -s hlsl -p ps_5_0 -f RenderPS -c Iceland --DXLocation "C:\Program Files (x86)\Windows Kits\8.1\bin\x86\d3dcompiler_47.dll" --isa c:\temp\myISA.isa

C:\Users\shaders\Render.hlsl

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Static Analyzer > CodeXLAnalyzer Command Line Interface > Using CodeXLAnalyzer Command Line Interface to compile OpenGL and Vulkan programs

Using CodeXLAnalyzer Command Line Interface to compile OpenGL and Vulkan programs

CodeXLAnalyzer.exe command line tool supports compilation and statistics generation OpenGL and Vulkan programs.

Details of available commands:

Details of available commands:		
-h	View available options	
	CodeXLAnalyzer.exe -h	
-S	Specify the source platform for the compilation.	
	CodeXLAnalyzer.exe -s opengl CodeXLAnalyzer.exe -s vulkan	
-S	View supported ASICS for DirectX (also-	
<platform> -</platform>	this is default ASICs list for compilation)	
1	CodeXLAnalyzer.exe -s opengl –l	
isa arg	Path to output ISA disassembly file(s).	
vert <arg></arg>	Full path to vertex shader's location	
tesc <arg></arg>	Full path to tessellation control shader's location	
tese <arg></arg>	Full path to tessellation evaluation shader's location	
geom	Full path to geometry shader's location	

<arg></arg>	
frag <arg></arg>	Full path to fragment shader's location
-c [asic] arg	By default, compilation will be done to all default available devices This option is repeatable.
	•
-a [analysis]	Path to the performance statistics output file. Requires compilation.
version	View the Catalyst driver version installed CodeXLAnalyzer_d.exe -s opengl –version
p	

Usage examples:

1. To build an OpenGL program with a vertex shader and a fragment shader attached and generate ISA

CodeXLAnalyzer.exe -s opengl --vert c:\shaders\glVertex.vert --geom c:\shaders\glGeom.geom --isa c:\output\myISA.txt

2. To build an Vulkan program with a vertex shader and a geometry shader attached, and generate ISA and performance statistics

CodeXLAnalyzer.exe -s vulkan --vert c:\shaders\vkVertex.glsl --geom c:\shaders\vkGeom.geom --isa c:\output\myISA.isa -a c:\output\myStats.txt

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Static Analyzer > CodeXLAnalyzer Command Line Interface > Generating and Interpreting CodeXLAnalyzer CLI's Live Register Analysis Report

Generating and Interpreting CodeXLAnalyzer CLI's Live Register Analysis Report

Using CodeXLAnalyzer CLI's live register analysis report, you can better understand the register usage of your HLSL shaders and OpenCL kernels throughout their execution. Live register analysis is a beta feature of CodeXLAnalyzer CLI, and it currently only fully supports HLSL shaders and partially supports OpenCL kernels.

Generating a live register analysis report for your kernel or shader:

As mentioned in the "Details of available commands" section above, in order to generate a live register analysis report, you need to make sure that your invocation command includes the following command line switches:

- 1. -- isa <arg> which instructs CodeXLAnalyzer to generate ISA disassembly for your kernel/shader
- 2. --livereg <arg> which instructs CodeXLAnalyzer to perform a live register analysis of the generated ISA disassembly

Usage examples:

CodeXLAnalyzer.exe -s cl -c Fiji --kernel DCT --isa c:\output\.isa --livereg c:\output\livereg.txt DCT_Kernels.cl

Let's break down the above command to understand its structure:

- 1. "-s cl" instructs CodeXLAnalyzer to work in OpenCL mode
- 2. "-c Fiji" sets Fiji as the target ASIC
- "--kernel DCT" sets DCT as the target kernel (this is the kernel to be analyzed; it is defined in DCT_Kernels.cl, which is the last argument in the above command)
- 4. "--isa c:\output\.isa" instructs CodeXLAnalyzer to generate an ISA disassembly file and save it in c:\output with a ".isa" file extension. The output file name is generated automatically.
- 5. "--livereg "c:\output\livereg.txt" instructs CodeXLAnalyzer to perform live register analysis, save the report in c:\output, and use "livereg.txt" as the report file name's suffix and extension.

After running the above command, we see the following output files in c:\output (our

destination folder): Fiji_DCT.isa Fiji_DCT_livereg.txt

The live register analysis report file is Fiji_DCT_livereg.txt.

For HLSL, the usage is similar:

CodeXLAnalyzer.exe -s hlsl -c Fiji -f VSMain -p vs_5_0 --isa c:\temp\.txt --livereg c:\temp\lreg.txt c:\temp\dx\BasicHLSL11_VS.hlsl

- 1. "-s hlsl" instructs CodeXLAnalyzer to work in HLSL mode
- 2. "-c Fiji" sets Fiji as the target ASIC
- 3. "-f VSMain" sets VSMain as the target shader
- 4. "--isa c:\output\.isa" instructs CodeXLAnalyzer to generate an ISA disassembly file and save it in c:\output with a ".isa" file extension. The output file name is generated automatically.
- 5. "--livereg "c:\output\livereg.txt" instructs CodeXLAnalyzer to perform live register analysis, save the report in c:\output, and use "livereg.txt" as the report file name's suffix and extension.

Report structure:

If you open up the live register analysis report file, you will see that it is a plain textual file. Each line in the file gives a snapshot of the register usage when the PC is at that specific ISA line. Each line in the report is of the following format:

e number> | <number of live registers> | <list of registers + access type> | <ISA instruction>

Where:

- 1. enumber> is the number of the current ISA disassembly line
- 2. <number of live registers> is the number of live registers when the PC is at that ISA line
- 3. st of registers + access type> is a list of n columns. Each column (except for the first one) refers to a register:
 - a. '^' indicates a register is written to
 - b. 'v' indicates a register is read
 - c. 'x' is used for a register which is written and read
 - d. ':' is used for register where the contents must be preserved across this instruction (live register)
 - e. A blank means that the register is not used
- 4. <ISA instruction> is the ISA disassembly of the relevant instruction

At the end of the report, you will find a summary in the

following format:

Maximum # VGPR used <Max VGPR used>, # VGPR allocated: <Number of VGPR allocated>

Where:

- 1. <Max VGPR used> is the number of VGPRs actually used throughout the code
- 2. <Number of VGPR allocated> is the number of VGPRs that were allocated

Two things to remember when inspecting the live register analysis report are:

- 1. If the number of live registers is lower than the number of allocated registers, it indicates that the SC could reduce VGPRs without spilling by introducing moves.
- 2. If registers have a very long liveness range without read/write access, those registers could be likely spilled at low cost.

Here is a sample live register analysis report:

- 1 | 9 | ::::::: :: | label_basic_block_1: s_swappc_b64 s[2:3], s[2:3]
- 2 | 9 | ::::::: | s_andn2_b32 s0, s9, 0x3fff0000
- 3 | 9 | ::::::: :: | s_mov_b32 s1, s0
- 4 | 9 | :::::: :: | s_mov_b32 s2, s10
- 5 | 9 | :::::: :: | s_mov_b32 s3, s11
- 6 | 9 | ::::::: :: | s_mov_b32 s0, s8
- 7 | 9 | ::::::: :: | s_buffer_load_dwordx8 s[4:11], s[0:3], 0x00
- 8 | 9 | ::::::: :: | s_buffer_load_dwordx8 s[12:19], s[0:3], 0x20
- 9 | 9 | ::::::::: | s_waitcnt lgkmcnt(0)
- 10 | 10 | ^ ::::v::: :: | v_mul_f32 v0, s7, v7
- 11 | 11 | :^ :::v:::: :: | v_mul_f32 v1, s11, v7
- 12 | 12 | ::^ :::v::: :: | v_mul_f32 v2, s15, v7
- 13 | 13 | :::^::v::: :: | v_mul_f32 v3, s19, v7
- 14 | 12 | x:::::v :::: | v_mac_f32 v0, s6, v6
- 15 | 12 | :x::::v ::: :: | v_mac_f32 v1, s10, v6
- 16 | 12 | ::x:::v ::: :: | v_mac_f32 v2, s14, v6
- 17 | 12 | :::x::v ::: :: | v_mac_f32 v3, s18, v6
- 18 | 11 | x::::v :::: | v_mac_f32 v0, s5, v5
- $19 \mid \ 11 \mid :x:::v \ ::::: \mid v_mac_f32 \ v1, \ s9, \ v5$

20 | 11 | ::x::v ::: :: | v_mac_f32 v2, s13, v5 21 | 11 | :::x:v ::: :: | v_mac_f32 v3, s17, v5 22 | 10 | x:::v :::::: | v_mac_f32 v0, s4, v4 23 | 10 | :x::v ::: :: | v_mac_f32 v1, s8, v4 24 | 10 | ::x:v ::: :: | v_mac_f32 v2, s12, v4 25 | 10 | :::xv ::: :: | v_mac_f32 v3, s16, v4 26 | 9 | vvvv :::: :: | exp pos0, v0, v1, v2, v3 27 | 5 | ::: :: | s_buffer_load_dwordx4 s[4:7], s[0:3], 0x40 28 | 5 | ::: :: | s buffer load dwordx4 s[8:11], s[0:3], 0x50 29 5 ::: :: | s_buffer_load_dwordx4 s[0:3], s[0:3], 0x60 30 | 5 | ::: :: | s_waitcnt expcnt(0) 31 | 6 | ^ ::v :: | v_mul_f32 v0, s6, v10 32 | 7 | :^ ::v :: | v_mul_f32 v1, s10, v10 33 | 8 | ::^ ::v :: | v_mul_f32 v2, s2, v10 34 | 7 | x:: :v :: | v_mac_f32 v0, s5, v9 35 | 7 | :x: :v ::: | v_mac_f32 v1, s9, v9 36 | 7 | ::x :v ::: | v_mac_f32 v2, s1, v9 37 | 6 | x:: v :: | v_mac_f32 v0, s4, v8 38 | 6 | :x: v :: | v_mac_f32 v1, s8, v8 39 | 6 | :::x v ::: | v_mac_f32 v2, s0, v8 40 | 6 | :::^ :: | v_mov_b32 v3, 1.0 :: | v_mov_b32 v4, 0 41 | 7 | ::::^ 42 | 7 | vvvv: :: | exp param0, v0, v1, v2, v3 vv | exp param1, v12, v13, v4, v3 43 | 4 | vv 44 | 0 | | s_endpgm

Maximum # VGPR used 13, # VGPR allocated: 14

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Power Profiler

Power Profiler

CodeXL's Power Profiler is a powerful tool to help analyze the energy efficiency of systems based on AMD APUs and majority of the recent dGPU(discrete GPU).

Features of the Power Profiler include:

- Report the following data:
 - Estimated average power consumed by APU and supported dGPU subcomponents.
 - Average frequency of the CPU cores and the internal GPU and supported dGPU.
 - o Thermal trend of the CPU compute-units and the internal GPU.
 - o Thermal trend of supported dGPU.
 - o CPU cores P-States.
- A command-line tool to for data collection and dump to text/binary format.
 - The CodeXL graphic client provides these Power Profiling capabilities:
 - Real-time monitoring
 - o Timeline view
 - o Summary view
 - Offline review of session data
 - Following hardware's are supported
 - o AMD APUs: Carrizo, Kaveri, Mullins, Temash, Stoney, Bristol
 - o AMD dGPUs: Graphics IP 7 GPUs, Radeon and FirePro models.

Installing the Power Profiler Linux Driver

Wider Linux power profiling support (DKMS)

Power Profiler's Performance Counters

Power Profiler Command Line Interface

Power Profiler Project Settings

Power Profiler Counters Selection Dialog

Power Profiler Session Tree

Power Profiler Timeline View

Power Profiler Summary View Power Profiler Session Control Remote Power Profiling AMDTPowerProfileAPI Library Limitations

Legal Disclaimer

The reports generated from this Software may include thermal and power estimates which may contain errors. The report values may deviate from the actual values and/or the published specifications. Therefore, these reports should not be relied upon for any commercial or production systems and the user is responsible for obtaining accurate thermal and power measurements.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Power Profiler > Installing the Power Profiler Linux Driver

Installing the Power Profiler Linux Driver

On Linux systems, the CodeXL Debian and RPM packages perform the driver installation automatically. However, if you've downloaded the CodeXL tar archive, you have to install the Power Profiler's Linux driver manually. This includes a simple step of running **AMDTPwrProfDriver.sh** script with root credentials.

Example: \$ tar -xf CodeXL_Linux_x86_64_2.0.XXXX.tar.gz \$ cd CodeXL_Linux_x86_64_2.0.XXXX \$ sudo ./AMDTPwrProfDriver.sh install

Installer will create a source tree for power profiler driver under /usr/src/ amdtPwrProf -<version number>. All the source files required for module compilation is located in this directory are under MIT license.

To uninstall the driver run the following command:

\$ \$ cd <codexl-install-dir> \$ sudo ./AMDTPwrProfDriver.sh uninstall

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Power Profiler > Wider Linux power profiling support (DKMS)

Wider Linux power profiling support (DKMS)

On Linux machine Power Profiler driver can also be installed with Dynamic Kernel Module Support (DKMS) framework support. DKMS framework automatically upgrades the power profiler driver module whenever there is a change in the existing kernel. This saves user from manually upgrading the power profiler driver module.

The DKMS package needs to be installed on target machines before running the installation steps mentioned in the above section. AMDTPwrProfDriver.sh installer script will automatically takes care of DKMS related configuration if DKMS package is installed in the target machine.

Example (for Ubuntu system): \$ sudo apt-get install dkms \$ tar -xf CodeXL_Linux_x86_64_2.0.XXXX.tar.gz \$ cd CodeXL_Linux_x86_64_2.0.XXXX \$ sudo ./AMDTPwrProfDriver.sh install

If the user upgrades the kernel version frequently it is recommended to use DKMS for installation.

^{© 2015} Advanced Micro Devices. All rights reserved.

CodeXL User Guide



Help > Using CodeXL > Power Profiler > Power Profiler's Performance Counters

Power Profiler's Performance Counters

Category	Name	Description	Notes
	Total APU Power	Average APU Power for the sampling period, reported in Watts. This is an estimated consumption value which is calculated based on APU activity levels.	
	CPU Compute Unit 0 Power CPU Compute Unit 1 Power	Average CPU Compute Unit Power for the sampling period, reported in Watts. This is an estimated consumption value which is calculated based on APU activity levels.	
		Average Integrated-GPU Power for the sampling period, reported in	

iGPU Power	Watts. This is an estimated consumption value which is calculated based on APU activity levels.	
PCIe- Controller Power	Average PCIe- Controller Power for the sampling period, reported in Watts. This is an estimated consumption value which is calculated based on APU activity levels. This value does not include the power consumed by PCIe devices connected to the PCIe bus.	
Memory- Controller Power	Average DDR Memory- Controller Power for the sampling period, reported in Watts. This is an estimated consumption value which is calculated based on APU activity levels. This value does not include	

Power

Display- Controller Power	the power consumed by the memory DIMMs. Average Display- Controller Power for the sampling period, reported in Watts. This value refers to the APU's internal display controller which may be used in notebook and embedded configurations. This is an estimated consumption value which is calculated based on APU activity levels. This value does not include the power consumed by the display.	
Cumulative APU Power	The accumulated energy consumed by the APU throughout the profile session. Reported in Joules.	Available only in the command line tool
Cumulative Compute Unit	The accumulated energy consumed by the CPU	Available

0 Power Cumulative Compute Unit 1 Power	Compute Unit throughout the profile session. Reported in Joules.	only in the command line tool
Cumulative iGPU Power	The accumulated energy consumed by the APU's Internal GPU throughout the profile session. Reported in Joules.	Available only in the command line tool
dGPU power	Average Discrete- GPU Power for the sampling period, reported in Watts. This is an estimated consumption value which is calculated based on dGPU activity levels.	The dGPU family name is prefixed with this counter name.
CPU Core 0 Average Frequency CPU Core 1 Average Frequency CPU Core 2 Average Frequency	Average CPU Core Frequency for the sampling period, reported in MHz. This is the Core Effective Frequency (CEF). The core can go into various P- States within the sampling period,	

	CPU Core 3 Average Frequency	each with its own frequency. The CEF is the average of the core frequencies over the sampling period.	
	iGPU Average Frequency	Average Integrated-GPU Frequency for the sampling period, reported in MHz.	
Frequency	dGPU Average Frequency	Average Discrete- GPU Frequency for the sampling period, reported in MHz.	The dGPU family name is prefixed with this counter name.
	CPU Core 0 Frequency Histogram CPU Core 1 Frequency Histogram CPU Core 2 Frequency Histogram CPU Core 3 Frequency Histogram	Histogram of CPU Core Effective Frequency (average frequency for the sampling period).	Available only in the command line tool
	iGPU Frequency Histogram	Histogram of Internal-GPU Effective Frequency (average	Available only in the command

		frequency for the sampling period)	line tool
	CPU Compute- Unit 0 Measured Temperature CPU Compute- Unit 1 Measured Temperature	sampling period). Measured CPU Compute Unit Average Temperature, reported in Celsius. The reported value is normalized and scaled, relative to the specific processor's maximum operating temperature. This value can be used to indicate rise and decline of	
Temperature	iGPU Measured Temperature	temperature. Measured Integrated-GPU Average Temperature, reported in Celsius. The reported value is normalized and scaled, relative to the specific processor's maximum operating temperature. This value can be used to indicate rise and decline of	

		temperature.	
	dGPU Measured Temperature	Measured Discrete-GPU Average Temperature, reported in Celsius. The reported value is normalized and scaled, relative to the specific processor's maximum operating temperature. This value can be used to indicate rise and decline of temperature.	The dGPU family name is prefixed with this counter name.
CPU Core State	CPU Core 0 P- State CPU Core 1 P- State CPU Core 2 P- State CPU Core 3 P- State	CPU Core P-State at the time when sampling was performed.	
Other counters	Core 0 Process ID Core 1 Process ID Core 2 Process ID Core 3 Process ID	Process Id of the process that was executed by the CPU core at the point in time when the core was sampled.	Available only in the command line tool

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Power Profiler > Power Profiler Command Line Interface

Power Profiler Command Line Interface

CodeXL Power Profiler provides a command line interface utility for users who prefer to use command interpreters like **cmd.exe** on Windows and **bash** on Linux. This CLI utility can be used to collect and analyze the profile data. It can also be used from a batch file or a test script.

Usage:

On Windows:

```
CodeXLPowerProfiler.exe <options>
```

On Linux:

```
CodeXLPowerProfiler <options>
```

The following options are supported:

	Specify what types of data will be collected during the profile session. The following data types are supported.
	power - collect all the available power counters
	temperature - collect all the available temperature counters
	frequency - collect all the available frequency counters
-P <profile options></profile 	cu_power - collect cpu compute-unit power counters
	cu_temperature - collect cpu compute-unit temperature counters
	gpu_power - collect gpu power counters

	gpu_temperature - collect gpu temperature counters
	core - collect core specific attributes. This includes core frequency, p-state and process id executing on the core.
	all - collect all the supported counters
	List all the counters supported by the local hardware, and the hardware devices that the profiler recognizes.
-1	The counter IDs can be used with the '-e' option.
	The hardware device IDs can be used with the '-D' option.
	Specify the comma separated list of counter names to be collected.
-e <counter,></counter,>	Use option '-l' to get the supported counter names.
	Note: use any one of the options -P or -e.
-D <counter,></counter,>	Specify the comma separated list of device ids. All the counters of these devices will be profiled and collected.
	Use option (-l) to get the supported devices.
-T <sampling interval></sampling 	Sampling interval in milli-seconds. The minimum value is 20ms.
-d <duration></duration>	Profile duration in seconds.
-o <path></path>	Specify the output file path. The default path will be %Temp%\Codexl- Power_ <timestamp> on Windows and /tmp/ Codexl-Power_<timestamp> on Linux.</timestamp></timestamp>
	Define the output file format: · csv - Comma Separated Value text file

-F <csv txt></csv txt>	txt - plain text file The default file format is CSV file.		
	Specify core affinity mask for the application to be launched. Default affinity mask is all the available CPU cores.		
-C <core mask=""></core>	0x1 = Core 0		
	0x2 = Core 1		
	0x4 = Core 2		
	0x8 = Core 3		
-b	Terminate the launched application after the specified profile duration.		
-w	Specify the working directory. Default will be the path of the launched application.		
-h	Displays this help information.		
-v	Print version string.		
-z <db file<br="">output dir></db>	Export results to a *.cxldb file which can be imported to CodeXL GUI application.		
	Process profiling is based on the IPC load. Collects power consumption of all running processes during their profile run. These power values may differ with the actual power consumption. These power values can be used to get a notion of power trend and relative power consumption among running processes.		
-M process	Note: Currently PMC counters are supported only on windows operating system. Supported AMD platforms are Kaveri, Mullins, Temash and Carrizo. Please refer limitation section for further details.		

Examples

- Collect power values for CPU Compute Units and GPU for the duration of 10 seconds, with sampling interval of 100 milliseconds:
- CodeXLPowerProfiler.exe –P cpu_power -P gpu_power -o c:\Temp\powerprof-out.txt –T 100 –d 10
- Collect all the supported counter values for the duration of 10 seconds with sampling interval of 100 milliseconds:
 - CodeXLPowerProfiler.exe –P all -o c:\Temp\powerprof-out.txt –T 100 –d 10
- Collect all the supported counter values for the duration of 300 seconds with sampling interval of 100 milliseconds, and output the data to a binary file that can be imported into the graphic client:
 - CodeXLPowerProfiler.exe P all C 0x3 -z c:\Temp\PowerOutput T 100 d 300
- Collect all running process with their energy consumption shares as well as all counters during a particular profile run. Where profile duration is set to 300 seconds and profile sampling period is set to 10 milliseconds.

CodeXLPowerProfiler.exe –M process –e all c:\Temp\PowerOutput –T 10 –d 300

Display help:

CodeXLPowerProfiler.exe -h

• Display version string:

CodeXLPowerProfiler.exe -v

Sample Text output files

Sample 1: Collecting non-cumulative counters

The following command collects all non-cumulative power counters for 10 seconds, sampling them every 500 milliseconds and dumping the results to a text file:

CodeXLPowerProfiler.exe -P power -d 10 -T 500 -o C:\temp\pwr_out.txt -F txt

When run on a Kaveri APU, the content of the result text file is as below:

CODEXL POWER PROFILE REPORT

PROFILE DETAILS

CPU Details:	Family(0x15) Model(48)
CPU Core Mask:	0xf
Sampling Interval:	500 milli-seconds
Profile Start Time:	Dec-09-2014_18-10-08
Profile Duration:	10 seconds

PROFILED COUNTERS

COUNT	ER ID NAME	CA	regory	UNIT	DESCRIPTION
0.	pcie-ctrl-power	Power	Watt	Average P	CIe-Controller Power for the sampling period,
1.	mem-ctrl-power	Power	Watt	Average	DDR Memory-Controller Power for the sampling period,
2.	total-apu-power	Power	Watt	Average A	APU Power for the sampling period, reported in Watts
4.	display-ctrl-power	Power	Watt	Average	Display-Controller Power for the sampling period,
5.	cpu-cu0-power	Power	Watt	Average	CPU Compute Unit Power for the sampling period,
16.	cpu-cu1-power	Power	Watt	Average	CPU Compute Unit Power for the sampling period,
27.	igpu-power	Power	Watt	Average In	ntegrated-GPU Power for the sampling period,

PROFILE RECORDS

RecordId Timestamp pcie-ctrl-power mem-ctrl-power total-apu-power display-ctrl-powe cpu-cu0-power cpu-cu1-power igpu-power

0	18:10:9:002	0.56	4.88	56.28	0.00	15.49	17.36	17.98
1	18:10:9:501	0.56	4.79	39.92	0.00	13.29	13.19	8.08
2	18:10:10:003	0.56	4.83	38.00	0.00	12.86	11.58	8.17
3	18:10:10:503	0.56	4.80	38.05	0.00	12.70	12.02	7.96
4	18:10:11:003	0.56	4.79	38.80	0.00	12.84	12.53	8.07
5	18:10:11:503	0.56	4.79	38.95	0.00	13.05	12.63	7.92
6	18:10:12:003	0.56	4.82	38.16	0.00	12.15	12.57	8.06
7	18:10:12:503	0.56	4.81	37.93	0.00	12.23	12.42	7.91
8	18:10:13:003	0.56	4.80	37.46	0.00	11.97	12.06	8.05
9	18:10:13:503	0.56	4.80	37.64	0.00	12.32	11.97	7.99
10	18:10:14:003	0.56	4.83	38.91	0.00	12.49	12.99	8.04
11	18:10:14:501	0.56	4.79	38.56	0.00	12.54	12.71	7.96
12	18:10:15:003	0.56	4.81	38.89	0.00	12.38	13.08	8.06
13	18:10:15:501	0.56	4.80	39.38	0.00	12.71	13.30	8.00
14	18:10:16:003	0.56	4.80	39.37	0.00	13.03	12.91	8.07
15	18:10:16:503	0.56	4.80	39.25	0.00	12.99	12.91	7.99
16	18:10:17:003	0.56	4.80	39.72	0.00	12.50	13.80	8.05
17	18:10:17:503	0.56	4.80	38.97	0.00	12.93	12.67	8.01
18	18:10:18:003	0.56	4.81	38.95	0.00	12.97	12.56	8.05
19	18:10:18:503	0.56	4.80	39.21	0.00	12.75	13.11	7.99
20	18:10:19:003	0.56	4.80	38.66	0.00	12.69	12.56	8.05

Sample 2: Collecting cumulative counters

The following command collects the cumulative power counters for 10 seconds and dumping the results to a text file:

CodeXLPowerProfiler.exe -d 10 -o c:\temp\cumulative.txt -e 3,6,17,28 –F txt

When run on a Kaveri APU, the content of the result text file is as below:

CODEXL POWER PROFILE REPORT

PROFILE DETAILS

CPU Details:	Family(0x15) Model(48)
CPU Core Mask:	0xf
Sampling Interval:	100 milli-seconds
Profile Start Time:	Dec-11-2014_15-57-44
Profile Duration:	10 seconds

PROFILED COUNTERS

COUNT	ER ID NAME	CATE	GORY	UNIT	DESCRIPTION
3.	total-apu-power-cuml	Power	Joule	Cumula	tive APU Power, reported in Joules.
6.	cpu-cu0-power-cuml	Power	Joule	Cumula	ative CPU Compute Unit Power, reported in Joules.
17.	cpu-cu1-power-cuml	Power	Joule	Cumul	ative CPU Compute Unit Power, reported in Joules.
28.	igpu-power-cuml	Power	Joule	Cumula	tive Integrated-GPU Power, reported in Joules.

CUMULATIVE COUNTERS

COUNTER	CUMULATIVE VALUE
total-apu-power-cuml	228.11
cpu-cu0-power-cuml	48.85
cpu-cu1-power-cuml	50.30
igpu-power-cuml	78.65

Sample 3: Collecting histogram counters

The following command collects the frequency histogram counters for 10 seconds and dumping the results to a text file:

CodeXLPowerProfiler.exe -d 10 -o c:\temp\histogram.txt -e 10,14,21,25,31 –F txt

When run on a Kaveri APU, the content of the result text file is as below:

CODEXL POWER PROFILE REPORT

PROFILE DETAILS CPU Details: Family(0x15) Model(48) CPU Core Mask: 0xf Sampling Interval:100 milli-secondsProfile Start Time:Dec-11-2014_16-00-35Profile Duration:10 seconds

PROFILED COUNTERS

COUNTE	R ID	NAME	CATEGO	DRY	UNIT	
10.	cpu-co	ore0-frequency-l	hist Frequency	MHz	z	Histo
14.	cpu-co	ore1-frequency-l	hist Frequency	MHz	z	Histo
21.	cpu-co	ore2-frequency-l	hist Frequency	MHz	Z	Histog
25.	cpu-co	ore3-frequency-l	hist Frequency	MHz	z	Histo
31.	igpu-fi	requency-hist	Frequency	MHz]	Histogra

DESCRIPTION

Histogram of CPU Core Effective Frequency. Histogram of CPU Core Effective Frequency. Histogram of CPU Core Effective Frequency. Histogram of CPU Core Effective Frequency.

HISTOGRAMS OF COUNTERS

COUN	ΓER	C	cpu-core0-frequency-hist
HISTOGRAM			
low	high	count	
0	200	0	
200	400	0	
400	600	0	
600	800	16	
800	1000	16	
1000	1200	5	
1200	1400	22	
1400	1600	29	
1600	1800	11	
1800	2000	1	
2000	2200	0	
2200	2400	0	
2400	2600	0	
2600	2800	0	
2800	3000	0	
3000	3200	0	
3200	3400	0	
3400	3600	0	
3600	3800	0	
3800	4000	0	

COUN	ΓER	cpu-core1-frequency-hist
HISTO	OGRAM	
low	high	count
0	200	0

200	400	0
400	600	0
600	800	6
800	1000	79
1000	1200	13
1200	1400	1
1400	1600	1
1600	1800	0
1800	2000	0
2000	2200	0
2200	2400	0
2400	2600	0
2600	2800	0
2800	3000	0
3000	3200	0
3200	3400	0
3400	3600	0
3600	3800	0
3800	4000	0

COUN	ΓER		cpu-core2
HISTO	OGRAM		
low	high	coun	t
0	200	0	
200	400	0	
400	600	0	
600	800	0	
800	1000	0	
1000	1200	0	
1200	1400	0	
1400	1600	2	
1600	1800	98	
1800	2000	0	
2000	2200	0	
2200	2400	0	
2400	2600	0	
2600	2800	0	
2800	3000	0	
3000	3200	0	
3200	3400	0	
3400	3600	0	
3600	3800	0	
3800	4000	0	

cpu-core2-frequency-hist

COUNTER			cpu-core3-frequency-hist
HISTOGRAM			
low	high	count	
0	200	0	
200	400	0	
400	600	0	
600	800	0	
800	1000	28	
1000	1200	45	
1200	1400	25	
1400	1600	1	
1600	1800	1	
1800	2000	0	
2000	2200	0	
2200	2400	0	
2400	2600	0	
2600	2800	0	
2800	3000	0	
3000	3200	0	
3200	3400	0	
3400	3600	0	
3600	3800	0	
3800	4000	0	
COUN	ΓER		igpu-frequency-hist
HISTO	OGRAM		
low	high	count	
0	100	0	
100	200	0	
200	300	0	
300	400	100	
400	500	0	
500	600	0	
600	700	0	
700	800	0	
800	900	0	
		-	

900 1000 0

Sample 3: Collecting process profiling data

The following command collects the process profiling data for 100 seconds and dumping the results to a text file:

CodeXLPowerProfiler.exe -d 10 –T 10 -o c:\temp\histogram.txt –M process –F txt

When run on a Carrizo APU, the content of the result text file is as below:

CODEXL POWER PROFILE REPORT

PROFILE DETAILS

CPU Details:	Family(0x15) Model(0x60)
CPU Core Mask:	0xf
Sampling Interval:	10 milli-seconds
Profile Start Time:	Mar-30-2016_00-20-44
Profile Duration:	100 seconds

PROCESS PROFILING DATA

SNoPIDSamples IPCPower(Joules)Power(%)NamePath

0 CodeX		11 rProfile			0.03 Program Files (x86)\CodeXL
1	0	38350		31755	0
95.28		[Syste:	m Proce	ess]	Unable
to read	l path				
2	648	1	0.38	0.01	0.01
lsass.e	xe				C:\Windows\System32
3	900	54	13.72	0.06	0.05
dwm.e	xe				C:\Windows\System32
4	4	224	96.76	0.24	0.21
System	ı				Unable to read path
5	2140	294	232.54		0.82
0.71		exploi	rer.exe		
C:\Win	dows				
6	760	4	1.04	0.00	0.00
svchos	t.exe				C:\Windows\System32

0.01 1648 16 5.71 0.02 7 MsMpEng.exe Unable to read path 0.01 2.21 548 15 0.01 8 Unable to read path csrss.exe 9 4396 1066 710.18 3.86 3.36 iexplore.exe C:\Program Files (x86)\Internet Explorer 10 178 170.15 680 0.29 svchost.exe 0.25 C:\Windows\System32 11 5296 1 0.44 0.00 0.00 conhost.exe C:\Windows\System32 12 0.00 2500 2 0.29 0.00 C:\Windows\System32 svchost.exe 2.57 0.01 13 1320 6 0.01 svchost.exe C:\Windows\System32 14 1160 0.80 0.00 0.00 2 SearchIndexer.exe C:\Windows\System32 15 15.70 0.04 456 35 0.03 C:\Windows\System32 svchost.exe 11 7.65 0.02 16 0.01 1108 C:\Windows\System32 svchost.exe 0.84 0.00 17 640 3 0.00 Unable to read path services.exe 1.00 0.00 18 936 2 0.00svchost.exe C:\Windows\System32 3 1.22 19 1448 0.00 0.00taskhostex.exe C:\Windows\System32 0.46 0.00 20 720 2 0.00C:\Windows\System32 svchost.exe 0.00 21 0.63 0.00 1612 2 WUDFHost.exe C:\Windows\System32 0.00 22 1296 4 1.42 0.00

spoolsv.exe 23 2736 1 0.25 0.00 sppsvc.exe C:\Windows\System32 0.00 Unable to read path

Profile Sesssion Power Consumption:11Total PID record collected 385440

115.00

© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Power Profiler > Power Profiler Project Settings

Power Profiler Project Settings

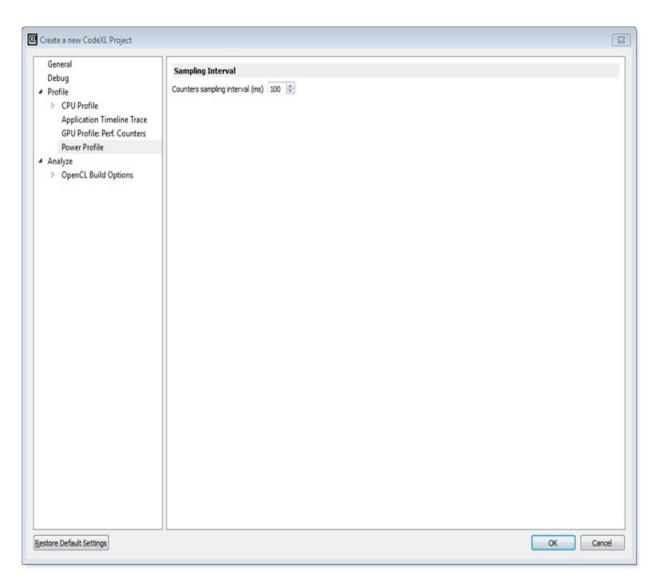
After switching to Profile mode, the Power Profiler Project Settings are accessible by any of these methods:

- In the Standalone application
 - o Pressing CTRL + P
 - o From the Profile menu, selecting "Profile Settings ... "
 - In Visual Studio

•

o From the CodeXL menu, selecting "CodeXL Project Settings ... "

The CodeXL Project Settings dialog appears. Navigate to the Power Profile sub-node under the Profile node in the options tree on the left.



The "Counters sampling interval (ms)" field sets the amount of time between every two consecutive sample operations of the profiler. The units used in this field are milliseconds. Setting an interval smaller than 100 milliseconds is discouraged as the overhead of the frequent sampling may skew the results and the design of the counters mechanism is not optimal for higher sample rates.

 $[\]ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.

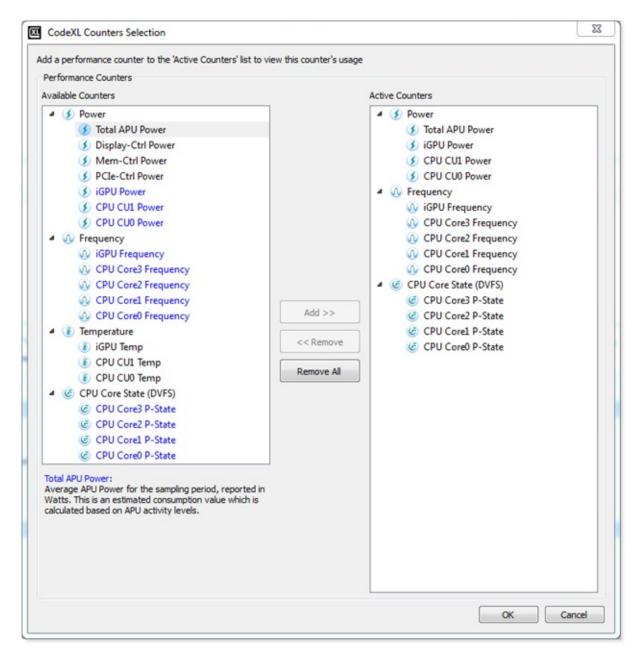




Help > Using CodeXL > Power Profiler > Power Profiler Counters Selection Dialog

Power Profiler Counters Selection Dialog

The counter selection dialog allows you to choose which performance counters will be monitored in the next power profiling session. The available counters are divided into 4 categories: Power, Frequency, Temperature, and CPU Core State. At the right side of the dialog window you will find the list of Active Counters. These are the counters which will be monitored in the next power profiling session. To get the description of a specific counter, click on it in the Available Counters pane at the left side of the dialog window.



Note: The Total APU Power counter is enabled by default, and cannot be deactivated. Except for the Total APU Power counter, the activation of all other counters is optional.

© 2015 Advanced Micro Devices. All rights reserved.



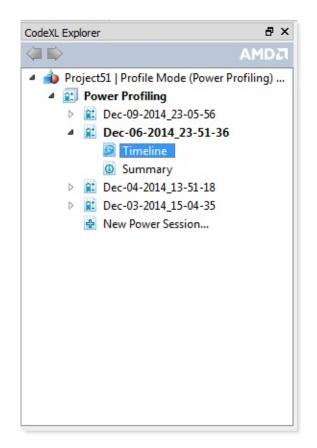


Help > Using CodeXL > Power Profiler > Power Profiler Session Tree

Power Profiler Session Tree

Navigating and displaying power profiling sessions is performed using the CodeXL Explorer tree, same as for other types of CodeXL profiling sessions. Each power profiling session is represented by a node in the tree, containing two sub-nodes for the Timeline and Summary views. Double clicking a node opens its respective view in the MDI space.

A session can be imported into the CodeXL project by rightclicking on the Power Profiling node and selecting 'Import Session'. The dialog that opens allows you to navigate to the location of the session to import, and displays a filtered list of files that correspond to the *.clxdb naming format.

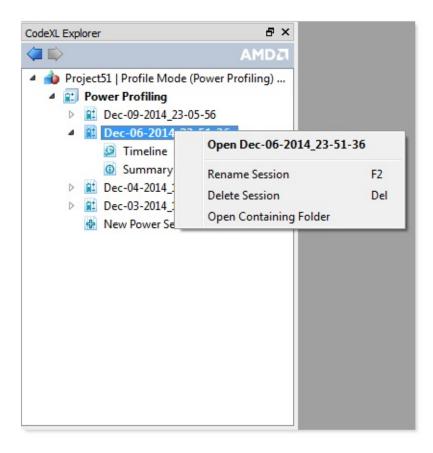


By clicking the "New Power Session..." node a new session is created, but sampling of data does not yet begin. The Timeline and Summary view tab will appear, and you can select which counters will be collected during the profile session by either

- Double-clicking the last line in the timeline charts legend table "Double-click to add or remove counters..."
- Selecting the "Select Power Profiling Counters..." command from the menu. This command is located in the standalone CodeXL app's Profile menu, and in the Visual Studio CodeXL menu.

To begin the sampling of data in the new session, click the "Start Profiling" toolbar button, or select "Start Profiling" from the Profile menu (standalone CodeXL) or CodeXL menu in Visual Studio.

Right-click on a session node in the explorer tree will display the session context menu which is identical to the context menu for other profile session types.



© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Power Profiler > Power Profiler Timeline View

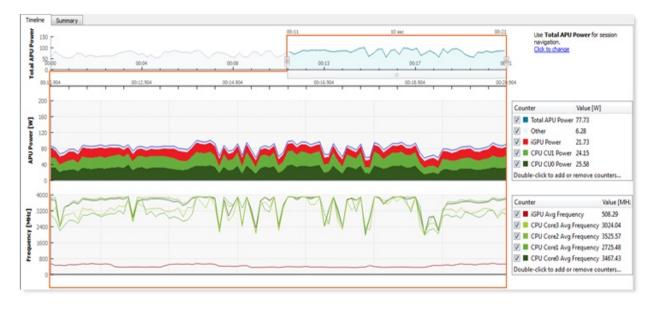
Power Profiler Timeline View

The Power Profiler Timeline View displays the measured values of the activated counters throughout the session. The horizontal axis of all charts represents the time that has passed since the beginning of the session. During profiling sessions, all of the charts in the Timeline View are being updated in realtime with the measured values which are streaming in. The uppermost ribbon (titled "Total APU Power") displays the overall power consumption of the APU throughout the session.

The top chart has an adjustable range slider that controls the display of all the other timeline charts. By performing such actions as dragging the slider sideways, extending or retracting it, you set the scope of attention and the focus of the timeline charts. Each of the charts below displays only the data that was collected in the time range corresponding to the slider's position and length. That is, the data in all timeline charts, except for the Total APU Power chart itself, is dictated by the time range which is selected by the Total APU Power chart's range slider.

Below the Total APU Power ribbon, you will find additional ribbons containing more graphs, according to the set of activated counters: A Power chart which displays the power consumed by specific APU components (such as CPU cores or integrated GPU), a Frequency chart which displays the frequency of the selected components, a Temperature chart which displays the thermal trend of the selected components, and a CPU State chart which displays the CPU core states. The APU Power graph is always displayed, since the Total APU Power counter is activated by default and cannot be deactivated. The other charts (frequency, temperature and CPU core state) are optional, and will only be displayed if the relevant counters were activated.

To the right side of each chart you will find a legend that displays the measured values at a specific point in time. To change the point in time for which the values are displayed, reposition the mouse cursor horizontally on one of the graphs. The list of counters in the legends is customizable, and specific counters can be removed/added between profile sessions.



© 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Power Profiler > Power Profiler Summary View

Power Profiler Summary View

The Power Profiler Summary View displays an analysis of the values measured throughout the session. Similarly to the Timeline View, this view is updated in real-time when power profiling sessions are running. At the upper-left side of the summary view, you can see the session duration which is the amount of time that the profiling session was in progress. The uppermost chart (titled "Power") shows the Total APU Power/Energy consumption. This chart has two modes: Cumulative and Average. The Cumulative mode displays the cumulative energy consumed by the APU components, measured in Joules. The Average mode displays the average power consumption of the APU components, measured in Watts. For each of these two modes, the Total APU Power Consumption value at the top of the summary view represents the value consumed by the whole APU.

If CPU or GPU frequency counters were activated for the session, you will find additional histogram graphs below the Power graph in the Summary View:

CPU Frequency Graph

This stacked histogram displays for each CPU core how much time it spent at each frequencies range.

GPU Frequency Graph

This histogram displays how much time the GPU spent at each frequencies range.



At the bottom of the Summary View you can find useful information about the current session:

- **Target Path:** The target application's path. In system-wide sessions, this field will be left empty.
- **Working directory:** the target application's working directory. In systemwide sessions, this field will be left empty.
- **Profile Session Directory:** the directory where CodeXL stored the session file.
- **Command Line Arguments:** the target application's command line arguments (if any). In system-wide sessions, this field will be left empty.
- **Environment Variables:** the target application's environment variables (if any). In system-wide sessions, this field will be left empty.
- **Profile Scope:** the scope of the current session (either system-wide, or system-wide with focus on an application).
- **Profile Session Type:** the type of this profiling session. This field will always be filled with the "Power" value for power profiling sessions.
- Profile Start Time: the time when the current session began.
- **Profile End Time:** the time when the current session ended.





Help > Using CodeXL > Power Profiler > Power Profiler Session Control

Power Profiler Session Control

The control of power profiling sessions is very simple:

- To start a power profiling session, open the Counters Selection Dialog (in the CodeXL menu, click on Profile->Select Power Profiling Counters...) and select the counters which will be monitored during the session. Click OK, then click the 'Start' button on CodeXL's session control toolbar.
- To stop a power profiling session, click the 'Stop' button on CodeXL's session control toolbar.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Power Profiler > Remote Power Profiling

Remote Power Profiling

For a detailed description of how to run a power profiling session on a remote machine, consult the Remote GPU Profiling, Power Profiling and Debugging section.





Help > Using CodeXL > Power Profiler > AMDTPowerProfileAPI Library

AMDTPowerProfileAPI Library

The CodeXL Power Profile API library is useful to analyze the energy efficiency of systems based on AMD CPUs, APUs and dGPUs (Discrete GPU). These APIs provide interface to read the power, thermal and frequency characteristics of APU/dGPU and their subcomponents.

These APIs are targeted for software developers who want to write their own application to sample the power counters based on their specific use case.

For detail information of these APIs, refer

AMDTPowerProfileAPI.chm (on Windows platforms) or **AMDTPowerProfileAPI.pdf** (on Linux based platforms) file in the **SDK/AMDTPowerProfile/doc** subdirectory under the CodeXL installation directory.

AMDTPowerProfileAPI shared library has dependencies on AMDTBaseTools and AMDTOSWrappers shared libraries, so the corresponding .DLL (on Windows system) and .SO (on Linux system) should be added.

To build and execute a test application (test.cpp) following steps should be performed on Linux machine.

- Assuming test.cpp is located at in /home/<user-dir>/samples
 \$ cd /home/<user-dir>/samples
- 2. Set LD_LIBRARY_PATH
 - \$ export LD_LIBRARY_PATH=<codexl-install-dir>
- 3. Compile application code

\$ g++ test.cpp –I<codexl-install-dir>/SDK/AMDTPowerProfile/inc –L<codexl-install-dir>/SDK/AMDTPowerProfile/bin/x86_64 – lAMDTPowerProfileAPI -L<codexl-install-dir> -lAMDTOSWrappers lAMDTBaseTools -o test

4. Execute

\$./test

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Power Profiler > Limitations

Limitations

- Multiple instance of CodeXL Power Profiler cannot be run simultaneously. If the CodeXL graphic client is conducting a power profile session then no other instance of the graphic client can perform a power profile and the command line tool cannot be run either, until the original profile session ends.
- ICELAND discrete GPU(Topaz-XT, Topaz PRO, Topaz XTL, Topaz LE) series is not support in 2.0 release.
- Please make sure you have latest catalyst driver installed before running power profiler. Newer version of discrete GPU may go to sleep (low power) state frequently if there is no activity in that GPU. In that case, power profiler may emits a warning AMDT_WARN_SMU_DISABLED. Counters may not be accessible during this state. It is advisable to bring discrete GPU to active state by running some openCL or openGL application, then run power profiling on that GPU.
- Process profiling is supported only with command line tool. If PMC (Performance Monitoring Counters) counters are not accessible and unable to calculate the IPC load, then compute unit power is distribute equally to each core. In that case power distribution to each process may not be accurate.





Help > Using CodeXL > Remote GPU Profiling, Power Profiling and GPU Debugging

Remote GPU Profiling, Power Profiling and GPU Debugging

CodeXL provides remote profiling and debugging capabilities. Using these features, you can execute GPU profiling, power profiling and debugging sessions of applications that run on a remote machine. This is useful for working with tablets and headless server units.

- <u>Running CodeXL Remote Agent</u>
- <u>The Agent's Configuration File</u>
- Performing Remote GPU Profiling
- Performing Remote Power Profiling
- Performing Remote GPU Debugging

^{© 2015} Advanced Micro Devices. All rights reserved.





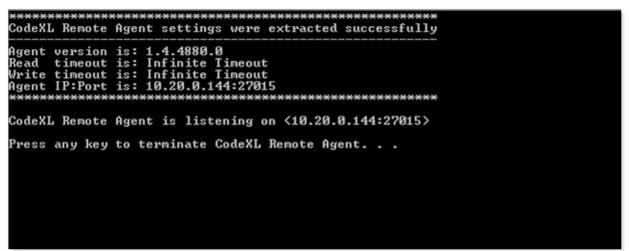
Help > Using CodeXL > Remote GPU Profiling, Power Profiling and GPU Debugging > Running CodeXL Remote Agent

Running CodeXL Remote Agent

As a first step, be sure that the remote machine has CodeXL Remote Agent installed. CodeXL Remote Agent ships with the CodeXL installer, and it is installed by default when installing CodeXL. You can also choose to install only CodeXL Remote Agent when using the installer.

CodeXL Remote Agent runs on the remote machine, and allows CodeXL clients located on other machines to connect the remote machine and execute GPU profiling and debugging sessions of applications on that machine.

When the remote is launched, it will output to the console a message in the following format:



In case the remote agent failed to launch successfully, it will output to the console a message describing the problem.

Please notice that by default the remote agent binds itself to the first valid IP address that it finds. In most cases, this

would be your desired behavior. However, if the remote machine has multiple IP addresses, and you would like to force CodeXL remote agent to bind itself to a particular address, you can use the --ip command line switch (--ip <ip_address>). For example, to force the remote agent to bind itself to 10.20.0.155, use the following command: On Windows: CodeXLRemoteAgent --ip 10.20.0.155 On Linux: CodeXLRemoteAgent-bin --ip 10.20.0.155





Help > Using CodeXL > Remote GPU Profiling, Power Profiling and GPU Debugging > The Agent's Configuration File

The Agent's Configuration File

CodeXL Remote Agent ships with a simple configuration file named CodeXLRemoteAgent.xml, which is located at the same folder as the CodeXLRemoteAgent executable. This configuration file defines several parameters which are being used by the agent. These parameters are being read by the agent just before it starts running. Therefore, if you change one of the values in the configuration file, you must rerun the agent for it to read the new values.

Here is a screenshot of the agent's configuration file:

</Settings>

Field	Description	
PortNumber	The port number on which the remote agent listens to incoming connections.	
ReadTimeoutMs	Read timeout to be used by the remote agent for incoming connections.	
WriteTimeoutMs	Write timeout to be used by the remote agent for incoming connections.	

Version	Version type, do not change this value.

Please pay special attention to the <PortNumber> parameter. This is the port on which the remote agent will listen to incoming connections. Prior to starting CodeXL remote Agent, you should verify that:

- 1. CodeXL Remote Agent is not blocked by the firewall on the remote machine.
- 2. The port number (27015 in the example above) is available (not being used by any other process), and that it is not blocked by a firewall. If there is a problem with the port, you can change this value and rerun the agent.

That's it for the remote machine. Now, go back to the local machine where the CodeXL application is installed. The following sections describe how to perform remote GPU Profiling and Debugging.





Help > Using CodeXL > Remote GPU Profiling, Power Profiling and GPU Debugging > Performing Remote GPU Profiling

Performing Remote GPU Profiling

1. Setting the remote target:

When you open the CodeXL project settings, look at the general tab: $% \label{eq:codeXL}$

🔟 CodeXL Project Settings		Х
General	General	
Debug		
Profile CPU Profile	CodeXL Project Name:	CodeXL Teapot@10.20.0.209
Application Timeline Trace GPU Profile: Perf. Counters	Target Host:	O Local Host Remote Host
	Remote Host Address:	10.20.0.209 Port: 27015 Test Connection
Power Profile	Target Application	
Frame Analysis Y Analyze	Application Type:	Desktop Application O Windows Store App
> Kernel/Shader Build Options	Executable Path:	C:\Users\Public\Documents\CodeXL\examples\Teapot\Release\AMDTTeaPot.exe
HLSL Build Options	Working Directory:	C:\Users\Public\Documents\CodeXL\examples\Teapot\Release
	Command Line Arguments:	
	Environment Variables:	
	Source Files	
	Source Files Directories:	tts(CodeXL(examples(Teapot(AMDTTeapot(src;C)(Users(Public(Documents(CodeXL(examples(Teapot(AMDTTeapotLib))))))))))))))))))))))))))))))))))))
	Source Files Resolution Root Directory:	
	analasi Sale de Central de S	
Restore Default Settings		OK Cancel

UI Control	Description	
	These two radio buttons determine whether	

Local Host	we are in a remote or local session. If Remote			
and Remote	Host is selected, all locations under the			
Host radio	Target Application box will refer to paths on			
buttons	the remote machine. Otherwise, all locations			
	will refer to paths on the local machine, as			
	usual.			
	The IP address of the remote machine (the			
	machine on which the remote agent is			
	running). In the above screenshot, the remote			
Remote Host	host is 10.20.0.144.			
text box	Alternatively, the host name of the remote			
	machine can be used instead of its IP			
	address, provided the name is recognized by			
	the network DNS.			
	The port (in the remote machine) on which			
	the remote agent is listening. Note that the			
Port text box	value of Port should match the PortNumber			
Fort text box	element value in			
	CodeXLRemoteAgentConfig.xml which is			
	located on the remote machine.			
	When this button is pressed, CodeXL will try			
	to connect to the remote agent running on			
	IP:Port (according to the text in the Remote			
	Host Address and Port text boxes). This			
Test	button can be used to verify that the remote			
Connection	agent is running on the remote machine and			
Button	that it is reachable by the CodeXL client.			
	Note that if the remote agent is not running			
	on the remote machine or if a firewall blocks			
	either CodeXL client or the remote agent –			
	the connection test would fail.			

2. For remote GPU performance counters profiling, have a look at the GPU Profile: Performance Counters tab (application trace settings act the same as in local sessions):

CodeXL Project Settings		Х
General Debug V Profile Application Timeline Trace GPU Profile: Perf. Counters Power Profile Frame Analysis V Analyze N Kernel/Shader Build Options HLSL Build Options	Profile Specific Kernels Profile only these kernels (names are case-sensitive, separated by semi-colons). Leave empty to profile all kernels.	Selection
Restore Default Settings	OK	Cancel

In a remote session, CodeXL does not know which counters are supported by the remote target's GPU.

Therefore, all possible HW families are available. In case that you picked a counter that is not supported by the remote host, you will see a blank column in the result spreadsheet (can be easily eliminated by using the "Hide zero columns" feature).

3. To launch Remote GPU profiling, click OK. Then, start profiling as usual.

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Using CodeXL > Remote GPU Profiling, Power Profiling and GPU Debugging > Performing Remote Power Profiling

Performing Remote Power Profiling

- 1. Set the remote target as for remote GPU profiling (you can find a description of how to set the remote target in section 1 under Performing Remote GPU Profiling above). Make sure that the remote agent is running and is reachable via network by CodeXL client. Also, make sure that the remote machine supports power profiling. See the CodeXL System Requirements section for a list of AMD devices that support power profiling.
- Open the Counter Selection Dialog and choose the desired set of counters, just like you do for local sessions. Note that a valid connection to the remote agent is required in order for CodeXL to populate the Counter Selection Dialog with the counters that are relevant to the specific remote machine.
- 3. Start the power profiling session by clicking the Start button, as you do for local sessions.





Help > Using CodeXL > Remote GPU Profiling, Power Profiling and GPU Debugging > Performing Remote GPU Debugging

Performing Remote GPU Debugging

1. Setting the remote debugging ports:

Go to Tools->Options, open the General tab and look at the Connection section. In the Remote Debugging Ports subsection, there are 4 ports which are being used by CodeXL on the local machine for remote debugging purposes. Please verify that the following requirements are fulfilled:

- a. All 4 port values are distinct.
- b. None of the 4 ports is blocked by a firewall.
- c. None of the 4 ports is being used by another process.

If necessary, you can change the port values to fulfill the above requirements.

2. Setting the remote target:

When you open the CodeXL project settings, look at the general tab:

Concerl		
General Debug	General	
✓ Profile	CodeXL Project Name:	CodeXL Teapot@10.20.0.209
> CPU Profile	Target Host:	O Local Host Remote Host
Application Timeline Trace GPU Profile: Perf. Counters	Remote Host Address:	10.20.0.209 Port: 27015 Test Connection
Power Profile Frame Analysis	Target Application	
/ Analyze	Application Type:	Desktop Application O Windows Store App
> Kernel/Shader Build Options	Executable Path:	C:\Users\Public\Documents\CodeXL\examples\Teapot\Release\AMDTTeaPot.exe
HLSL Build Options	Working Directory:	C:\Users\Public\Documents\CodeXL\examples\Teapot\Release
	Command Line Arguments:	
	Environment Variables:	
	Source Files	
	Source Files Directories: Source Files Resolution Root Directory:	<pre>its\CodeXL\examples\Teapot\AMDTTeapot\src;C:\Users\Public\Documents\CodeXL\examples\Teapot\AMDTTeapotLib;</pre>

UI Control	Description

Local Host and Remote Host radio buttons	These two radio buttons determine whether we are in a remote or local session. If Remote Host is selected, all locations under the Target Application box will refer to paths on the remote machine. Otherwise, all locations will refer to paths on the local machine, as usual.
Remote Host text box	The IP address of the remote machine (the machine on which the remote agent is running). In the above screenshot, the remote host is 10.20.0.144. Alternatively, the host name of the remote machine can be used instead of its IP address, provided the name is recognized by the network DNS.
Port text box	The port (in the remote machine) on which the remote agent is listening. Note that the value of Port should match the PortNumber element value in CodeXLRemoteAgentConfig.xml which is located on the remote machine.

3. Click OK, and start debugging as usual.

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Glossary > Compute or Graphic Memory Allocated Object

Compute or Graphic Memory Allocated Object

A compute or graphic memory allocated object is an OpenCL or OpenGL object created by a computer application, which is allocated on the memory of the OpenCL or OpenGL device.

An allocated object can reside in the computer's main memory (as part of the driver's used virtual memory), in the device's internal memory, or in both. In multi-device OpenCL contexts and multi-renderer OpenGL contexts, an object's data can even reside in multiple devices at the same time. CodeXL currently supports the following allocated object types.

- · OpenCL compute contexts
- · OpenCL command queues
- OpenCL command events*
- · OpenCL Image objects
- · OpenCL buffer objects
- · OpenCL computation programs
- · OpenCL kernel objects*
- OpenCL sampler objects*
- OpenGL render contexts
- · OpenGL render contexts' static buffers
- · OpenGL texture objects
- · OpenGL render buffers
- OpenGL framebuffer objects (FBOs)*
- · OpenGL vertex buffer objects (VBOs)
- OpenGL shading programs*
- OpenGL shaders
- · OpenGL display lists
- · OpenGL pixel buffer objects (PBuffers)*

The memory consumption of these objects is insignificant, but their creation and deletion is still monitored by CodeXL. **Function Types:** OpenCL functions can be divided into groups by their effect on the OpenCL implementation or by their outputs. Note that an OpenCL function can belong to more than one of these groups, or to none at all:

- Get Functions retrieve information from the OpenCL ICD.
- Buffer and Image Functions are functions related to buffer and image objects.
- Program and Kernel Functions are related to computation programs and kernels.
- Queue Functions are related to command queue objects.
- Synchronization Functions perform synchronization operations.

OpenGL functions can be grouped by their effect on the OpenGL implementation or by their outputs. Note that an OpenGL function can belong to more than one of these groups, or to none at all:

- Get Functions retrieve information from the OpenGL driver.
- State Change Functions change the values of OpenGL State Variables.
- Draw Functions can have a visible effect on the draw buffer.
- Raster Functions copy pixels from or to frame buffers.
- Programs and Shaders Functions are related to shaders and shading programs.
- Texture Functions create, delete, or manipulate textures.
- Matrix Functions are related to the matrix stacks (modelview matrix, projection matrix, color matrix, etc.).
- Name Functions are related to the OpenGL selection mode.
- Query Functions are related to OpenGL query items.
- Buffer Functions are related to OpenGL buffer objects.

To find out what groups a given function belongs to, look for its entry in the Function Calls Statistics tab (in the CodeXL Statistics view). The information is displayed under the "Function Type" column. The Statistics view's Function Types Statistics tab shows details of your API usage by these groups, as well as more detailed information about the use of each function type.





Help > Glossary > Frames and Frame Terminators

Frames and Frame Terminators

Computation Frame: A computation frame is a set of OpenCL API calls, typically the largest set of calls an OpenCL compute context performs that can be considered a single logical operation. A computation frame is comparable to a render frame in a graphics engine. It is recommended for applications rendering graphics to define the computation frame as the set of commands used to calculate the data for a single render frame. Having a notion and boundary of what comprises a frame permits measurements such as frame times and frame rates, as well as API call statistics, which are useful in debugging and profiling.

Frame Terminator: Frame terminators are the functions that end your application computation and render frames. CodeXL uses frame terminators for frame-per-second calculations, statistics analysis, and other measurements, as well as for determining when to stop the execution after you press the Step Out button (Shift+F11). When choosing frame terminators for your project, ensure that at least one of the functions you choose is called for each frame you render. If you are using the OpenGL debug engine, you must select at least one render (OpenGL) frame terminator. A typical selection of Frame Terminators is clFinish for OpenCL and the SwapBuffers option for OpenGL.





Help > Glossary > Vertex Batch

Vertex Batch

Vertex Batch: A vertex batch is a group of vertices sent to the graphic driver in a single API call. In OpenGL, this effectively means the number of vertices sent to the graphic pipeline with a single function call. For example, the function glVertex3f always generates a 1-vertex batch, while glDrawElements creates a batch of the size determined by its count parameter value. The number of vertex batches sent to the driver represents the **cost** of drawing, while the batch size is the **benefit** of each function call. Drawing 200 1-vertex batches has the same benefit as drawing a single 200-vertex batch, for 200 times the cost. For this reason, it is best to draw as many vertices using as few batches as possible.





Help > Bundled Sample Applications

Bundled Sample Applications

CodeXL contains several bundled sample applications to demonstrate CodeXL usage scenarios and features.

The bundled sample applications are:

- <u>Teapot</u>
- Matrix Multiply
- **D3DMultiThreading**

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.



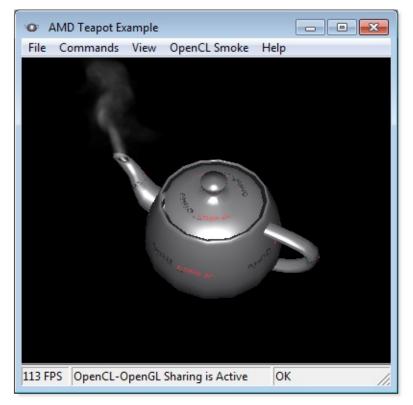


Help > Bundled Sample Applications > Teapot

Teapot

The Teapot sample application draws a spinning teapot with steam coming out of its spout. The steam vapors movement is simulated using OpenCL. The rendering of the teapot and steam is implemented using OpenGL. In the Linux version of Teapot, the window, menu bar and UI widgets are implemented using the <u>FLTK</u> library.

The Teapot application can be used to demonstrate the features of CodeXL's GPU Debugger, GPU Profiler and Static Analyzer.



Building the Teapot sample application

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Bundled Sample Applications > Teapot > Building the Teapot sample application

Building the Teapot sample application

CodeXL includes a pre-built binary of the Teapot application. Follow the steps below if you wish to build it yourself.

On Windows

Follow these steps to build AMDTTeapot on Windows from the default install location:

- · Launch Visual Studio with administrator privileges.
- From the CodeXL menu, select the 'Open Teapot Sample Project' command.
- Wait for the project to open and press F7.

On Linux

Follow these steps to build AMDTTeapot on Linux:

- Install the FLTK library
 - The bundled binaries of AMD Teapot were built with 64-bit binaries of FLTK 1.1.10.
 - o Download FLTK sources from http://www.fltk.org
 - Extract and build the fltk libraries from the source files, then install the created fltk libraries
- Modifying The AMDTTeaPot Makefile
 - There are two makefiles in the Teapot sample folder. One is the /examples/Teapot/AMDTTeaPot/Makefile which you'll need to edit, and the other is the /examples/Teapot/AMDTTeaPotLib/Makefile that does not need to be changed.

- o Open /examples/Teapot/AMDTTeaPot/Makefile in a text editor.
- Replace the -L"Replace with path to your local FLTK lib folder" with the path of your local FLTK libraries, for example -L/usr/lib64 in case you placed the libraries in the system folder and you are running a 64bit system
- Replace the -I"*Replace with path to your local FLTK headers folder*" with the path of your local FLTK include files. Please note that the FLTK headers are placed in a folder named "FL" and the path you provide should point to the parent of the "FL" folder. Example -I/user/include if you placed them in the system include and not -I/user/include/FL
- Building the Teapot
 - Teapot consists of a library and an application so first the library needs to be built.
 - o cd to the /examples/Teapot/AMDTTeaPotLib folder
 - o make all
 - o if everything went well you'll see libAMDTTeaPot.a in that folder
 - o Now the main application can be built.
 - o cd to /examples/Teapot/AMDTTeaPot
 - o make all
 - If successful the output should be in /examples/Teapot/release. Look for a file named AMDTTeaPot-bin.

Note that CodeXL RPM and Debian packages install CodeXL and is bundled sample applications under the /opt folder, which requires elevated privileges to write to on some Linux distributions. $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.



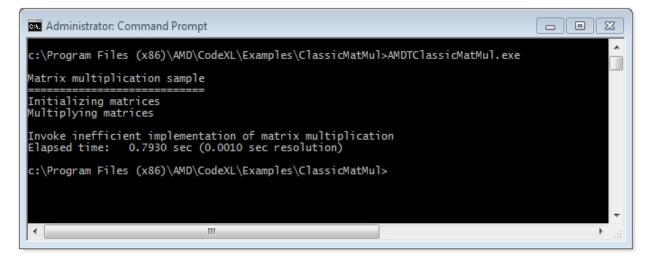


Help > Bundled Sample Applications > Matrix Multiply

Matrix Multiply

The matrix multiplication sample application performs multiplication of two matrices using 3 different implementations. All implementations are using ANSI C code.

The sample takes a command line argument that defines which of the matrix multiplication implementations will be executed. See the following table for descriptions of the 3 implementations and the command line argument that invokes each one.



This sample demonstrates how the CPU Profiler can be used to detect bottlenecks, identify problematic memory access patterns, and verify improved performance.

To select which implementation of the matrix multiplication will be performed, open the CodeXL Project Settings dialog, or in Visual Studio open the VS Project Settings dialog.

#	Multiplication Implementation	Command line argument	Description
1.	Inefficient		Inefficient implementation that performs redundant loop iterations.
2.	Classic	-C	Classic textbook implementation that uses naïve nested loops. The loops perform a sub-optimal memory access pattern.
3.	Improved	-i	Improved implementation that uses nested loops with continuous memory access pattern.

CodeXL Project Settings		Х
Debug	General	
> CPU Profile Ta Application Timeline Trace GPU Profile: Perf. Counters	odeXL Project Name: arget Host: emote Host Address:	AMDTClassicMatMul
Frame Analysis	arget Application	Desktop Application Windows Store App
Kernel/Shader Build Options Ex HLSL Build Options	xecutable Path: /orking Directory:	C: \ProgramData \AMD\CodeXL \Examples \ClassicMatMul\Release \AMDTClassicMatMul.exe
	ommand Line Arguments: nvironment Variables:	<mark>د</mark>
	iource Files	
	burce Files Directories: burce Files Resolution oot Directory:	C: \ProgramData\AMD\CodeXL\Examples\ClassicMatMul\Release
Restore Default Settings		OK Cancel

Project Settings for launching the classic implementation of matrix multiplication using the -c command line argument

Use Time-Based Profiling to detect the redundant loop iterations in the inefficient implementation (launched by not

supplying any command line argument).

Aug-12-2015_01-42-29 (CPU: Time-based Sampling)										
Profile Overview										
5 Hottest Functions										
Function	Samples	% of Hotspot Samples	Module							
inefficient_multiply_matrices(void)	1,228	98.56%	AMDTClassicMatMul.exe							
initialize_matrices(void)	18	1.44%	AMDTClassicMatMul.exe							

Double clicking the inefficient_multiply_matrices(void) opens the source code view which demonstrates why this function is not efficient:

ð		Aug-12-2015_01-42-29 (CPU: Time-based Sampling)	×			
Profile	Profile Overview II:cMatMul\AMDTClassidMatMul.exe - Source/Disassembly II					
c: \jenkins	workspace (co	dexl-1.8\release\codexl\1.8\codexl\examples\amdtclassicmatmul\amdtclassicmatmul.cpp				
Function	[0x11412e	0 - 0x1141416] : inefficient_multiply_matrices(void) Display: <u>All Data, System Modules Hidden</u>				
Line	Address	Source Code	Code Bytes	Hotspot Samples		
118		}				
119		}				
120						
121		void inefficient_multiply_matrices()				
▶ 122	0x11412e0	1				
123		// !!! Catch me if you can - doing some stupid things !!!				
▷ 124	0x11412e8	for (int I = 0; I<3; I++)				
125						
▷ 126	0x11412f0	classic_multiply_matrices();		1,228		
127		}				
▷ 128	0x1141412	}				

Use Event-Based Profiling with the Assess Performance session type to diagnose the problems in the classic implementation (launched by supplying "-c" as the command line argument):

A		Aug-12-2015_01-48-01 (CPU: Ass	ess Performance)			×						
0 Profile	Overview	📴cMatMul\AMDTClassicMatMul.exe - Source/Disassembly 🛛										
c: \jenkins	workspace (co	dexl-1.8\release\codexl\1.8\codexl\examples\amdtdassicmatmul\a	ndtclassicmatmul.cpp									
Function	[0x1081070) - 0x108118f] : classic_multiply_matrices(void) 🔻 Display: Al D	ata, System Modules Hidde	<u>en</u>							_	
Line	Address	Source Code Code B	/tes Hotspot Samples	% of Hotspot Samples	DC accesses	Misalign access	CPU clocks	Ret inst	Ret branch	Ret misp branch	DC misses	TLB miss
89		void classic_multiply_matrices()										
♦ 90	0x1081070	{										
91		// Multiply the two matrices										
Þ 92	0x1081078	for (int i = 0 ; i < ROWS ; i++)										
93		1										
Þ 94	0x1081080	for (int j = 0 ; j < COLUMNS ; j++)										
95		{										
♦ <mark>96</mark>	0x1081096	float sum = 0.0 ;	8	0.34%			4		1		8	
97		for (int $k = 0$; $k < COLUMNS$; $k++$)										
98		{										
Þ 99	0x10810a2	sum = sum + matrix_a[i][k] * matrix_b[k][2,361	99.66%	1,041		1,342	1,752	553		2,361	
100		}									-	
▷ 101	0x108115a	matrix_r[i][j] = sum ;					28	42	16	4		
102		}										
103		}										
▷ 104	0x108118b	}										

Large number of cache misses in the internal nested loop non-sequential access to arrays

Finally, executing the improved implementation (launched by supplying "-i" as the command line argument) shows the number of cache misses is significantly reduced:

A		Aug-12-2015_01-48-01 (CPU: Assess Performance)	×	A)	Aug-12-2015_01-53-5	1 (CPU: Assess	Performance)		×			
0 Profile	e Overview	📴cMatMul/AMDTClassicMatMul.exe - Source/Disassembly 🗵										
c:\jenkins	workspace (co	dexi-1.8\release\codexi\1.8\codexi\examples\amdtclassicmatmul\amdtclass	icmatmul.cpp									
Function	[0x108119) - Ox10812d5] : improved_multiply_matrices(void) 🔻 Display: All Data, :	System Modules	Hidden								Hots
Line	Address	Source Code	Code Bytes	Hotspot Samples	% of Hotspot Samples	DC accesses	Misalign access	CPU clocks	Ret inst	Ret branch	Ret misp branch	DC misses
104		1										
105		void improved_multiply_matrices()										
	0x1081190											
108		// Multiply the two matrices										
▷ 109	0x1081197	for (int i = 0 ; i < ROWS ; i++)										
110		{										
◊ 111	0x10811b0	for (int k = 0; k < COLUMNS; k++)					1	3	5	2		
112		{										
▷ 113	0x10811d0	for (int j = 0 ; j < COLUMNS ; j++)		1	2.22%			4	3	8	2	1
114						1						
	0x10811f0	matrix_r(i)[j] = matrix_r(i)[j] + matrix_a(i)[k] * matrix_b(k][j]	;	44	97.78%	407		248	909	654	4	44
116												_
117 118												
	0x10812d1	1										

Building the Matrix Multiply sample application





Help > Bundled Sample Applications > Matrix Multiply > Building the Matrix Multiply sample application

Building the Matrix Multiply sample application

CodeXL includes a pre-built binary of the Matrix Multiply application. Follow the steps below if you wish to build it yourself.

On Windows

Follow these steps to build Matrix Multiply on Windows from the default install location:

- · Launch Visual Studio with administrator privileges.
- From the CodeXL menu, select the 'Open Matrix Multiply Sample Project' command.
- Wait for the project to open and press F7.

On Linux

Follow these steps to build Matrix Multiply on Linux:

- Navigate to codexl_folder_path/examples/ClassicMatMul/src
 - o codexl_folder_path is the folder in which you installed or unzipped CodeXL
- Type 'make' and hit Enter.

Note that CodeXL RPM and Debian packages install CodeXL and is bundled sample applications under the /opt folder, which requires elevated privileges to write to on some Linux distributions.





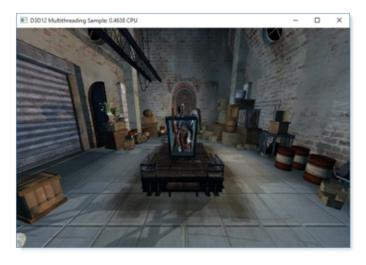
Help > Bundled Sample Applications > D3D12Multithreading

D3D12Multithreading

The D3D12Multithreading is a sample from the Microsoft D3D SDK, which is bundled with CodeXL.

The sample demonstrates the use of multiple thread with Direct3D 12.

The D3D12Multithreading sample can be used to get to know the Frame Analysis feature in CodeXL.



© 2015 Advanced Micro Devices. All rights reserved.





Help > Bundled Sample Applications > D3D12Multithreading > Building the D3D12Multithreading sample application

Building the D3D12Multithreading sample application

CodeXL includes a pre-built binary of the D3D12Multithreading application. Follow the steps below if you wish to build it yourself.

The D3D12Multithreading sample is a windows only sample, and can only be built in Visual Studio 2015, with the Windows 10 SDK installed.

On Windows

Follow these steps to build D3D12Multithreading on Windows from the default install location:

- Make sure that you have a Visual Studio 2015 IDE installed.
- \cdot Make sure that the Windows 10 SDK is installed on your system.
- · Launch Visual Studio 2015
- From the CodeXL menu, select the 'Open D3D12Multithreading Sample Project' command.
- Wait for the project to open and press F7.

 $[\]ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials

Tutorials

<u>GPU Debugger Tutorial</u> <u>CPU Profiler Tutorial</u> <u>GPU Profiler Tutorial</u> <u>Static Kernel Analyzer Tutorial</u>

© 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > GPU Debugger Tutorial

GPU Debugger Tutorial

<u>Host Debugging</u> <u>API-Level Debugging</u> <u>Kernel Debugging</u> <u>Additional Information</u>

© 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > GPU Debugger Tutorial > Host Debugging

Host Debugging

Host Debugging is currently only available in the following configurations:

1. Linux CodeXL Standalone application

2. Windows CodeXL Visual Studio extension [native (C/C++)

32-bit target apps only]

Other configurations support only API-Level debugging.

In the supported configurations, CodeXL allows Host debugging of any application, regardless of any APIs used.

The following features are supported:

- 1. Host source code breakpoints
- 2. Host stepping (step-in, step-over, step-out)
- 3. Host break
- 4. Host locals display
- 5. Host expression evaluation (watch window)

CodeXL provides information directly from the system's native debugger, supporting any code and debug information that is available as those debuggers.

In the CodeXL standalone application, the debug controls are located in the Execution toolbar, next to the "Start Debugging" and "Stop Debugging" buttons. Setting source code breakpoints is as simple as opening a source file (Ctrl+O) and clicking the margins (F9). The Locals and Watch views are shared between Host Debugging and Kernel Debugging for a seamless experience.

In the CodeXL Visual Studio extension, Visual Studio's built-in views are used, providing a debugging experience as close to Visual Studio's built-in debug engines as possible.

 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > GPU Debugger Tutorial > API-Level Debugging

API-Level Debugging

CodeXL allows API-level debugging of any OpenCL and / or OpenGL based application, regardless of implementation version and underlying hardware.

API Debugging Basics

CodeXL API-level debugging is how CodeXL displays the CPU (client) side of your OpenCL or OpenGL application. This means CodeXL can suspend the debugged application at any OpenCL and OpenGL API function, display the call stack for these function calls, display API calls history and give many analysis options for debugging this part of the application.

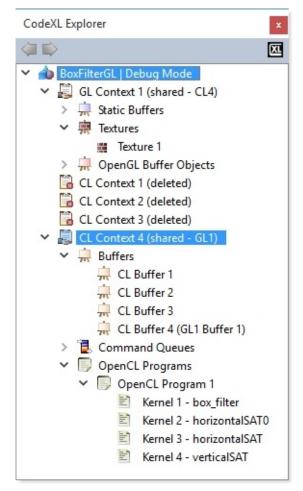
To run the debugged application, press the Go button (F5).

Once the application is running, you can suspend it by adding an API breakpoint or by pressing the "break" button.

Once the application is suspended, you can use the API steps drop-down to choose between API Step (Ctrl + F10 - run to the next API function call), Draw Step (Ctrl + Shift + F10 - run to the next OpenGL draw call) and Frame Step (Ctrl + F11 - run to the next frame terminator API function call).

The CodeXL Explorer tree view is an invaluable view in CodeXL. When the debugged application is suspended (by a breakpoint, a step command, a break command, etc.), this view fills with the hierarchy of all OpenCL and OpenGL objects allocated by the debugged application. Each context or category can be expanded to show its contained objects, and the CodeXL Properties view updates with the details of the selected object. Double-clicking any of the objects usually shows the contents of the object in some form.

CodeXL Explorer

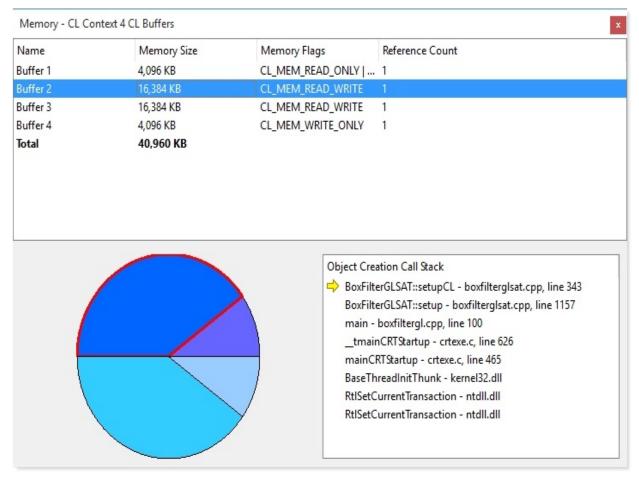


View Compute and Graphic Memory Consumption

All of the compute and graphic memory information collected by CodeXL is displayed in the Memory view. To open this view, right-click on the object in the object tree, and select the "Memory analysis" option of the object. This view lists all the allocated objects of a given type (for example, context 1's buffers), as well as information regarding their memory consumption depending on the kind of objects listed. The graph view shows a breakdown of all the graphic memory consumed by the items in the current category. To see the scenario that led to an object's creation, use the Creation Calls Stack view. If the debugged application has debug information, doubleclicking on a line in the Creation Calls Stack view opens the

location in the source file.

Memory View



Find Unrecommended Function Calls

Open the Statistics view. In the "Function Calls Statistics" tab, you might see warning icons next to some of the function names. Click any of these functions to display an explanation of why is it unrecommended, and what alternative is available.

Statistics View

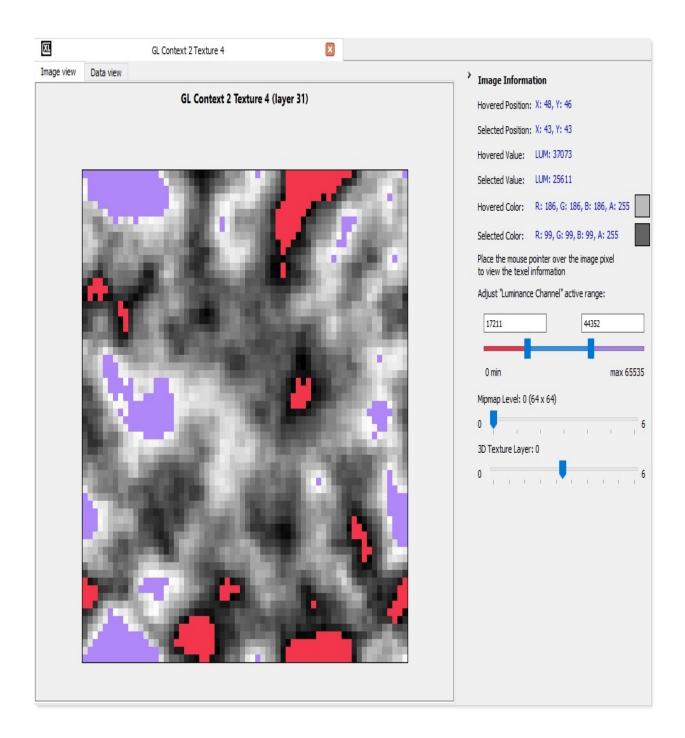
<u>u</u> <u>Ju</u>		
Function Types Function Calls	Deprecated Function C	alls History Vertex Batches
unction Name	% # of Calls	Function type /
glTexCoord2f	12.25 2,076	Texture Deprecated
glVertex3f	12.25 2,076	Draw Deprecated
glMatrixMode	9.19 1,558	State Change Matrix Deprecated
🖞 wglGetProcAddress	7.97 1,351	Get
glLoadldentity	6.13 1,039	Matrix Deprecated
glBindBuffer - GL_PIXEL_UN	6.13 1,039	State Change Buffer
glBindTexture - GL_TEXTUR	3.08 522	Texture
glDisable - GL_DEPTH_TEST	3.07 521	State Change
🛕 glBegin - GL QUADS	3.07 520	Draw Deprecated
glDisable - glDisable - GL_DEP	TH_TEST 520	State Change
alDicable GL TEVTURE 2D	2 07 520	Qate Change
		 glBegin - GL_QUADS Immediate mode rendering (glBegin-glEnd blocks) is considered an inefficient way to push primitive data into the graphic system. This mode requires making a large amount of function ca for describing the primitives' data. Instead, consider using VBOs (Vertex Buffer Objects) or Vertex Arrays, which enable pushir large data chunks at once.

x

View Buffers, Images, and Textures Information and Data

Double-clicking the main branch for buffers, images, or textures in the CodeXL Explorer (such as "CL Context 1 Images" or "GL Context 3 VBOs") opens a thumbnail view for those objects, to give a quick overview. Clicking an object in this view or double-clicking it in the CodeXL Explorer opens an Object view for it in the main views area. This view has a Data view spreadsheet showing the raw object data where each cell in the spreadsheet shows the value of each pixel, and an Image view that visualizes the object. Clicking on a pixel samples it and keep its information in the "Selected Pixel" entry in the Information Panel on the right section of the view. Using the Images and Buffers toolbar, you can disable some or all the channels and zoom in and out on the image. You also can invert the image's colors, convert it to grayscale and rotate it. The Image Information panel on the right gives you controls over multi-imaged or multi-layered objects, such as 3D images, array textures, or mipmapped textures. Use the various sliders to change between the object elements.

Objects View



OpenCL buffers and OpenGL VBOs do not have the context of pixels, coordinates, and channels; thus, they are not visualized by the image view. Since they contain raw byte data, they even have no type or format information built-in into them. Instead, the Information panel has a combo-box letting you choose from the available data types and formats for use in their respective

XL	clutil.hpp	X	CL Context 4 CL Buffer	r 1 🔣 🕅	CL Context 4 CL Buffer 2	×	
ata view						Buffer Display	Ontions
				G	id Zoom:	Type:	
	adecimal values						
_ Snow nex	adecimal values					Format:	int4
	i	i	i	i	^	Offset:	0
0	61	92	113	255		Stride:	0
1	113	173	210	510			0
2	173	260	308	765			
3	242	350	415	1020			
4	313	441	533	1275			
5	385	536	659	1530			
6	452	631	778	1785			
7	508	719	881	2040			
8	581	813	994	2295			
9	646	899	1099	2550			
10	720	994	1213	2805			
11	803	1098	1336	3060			
12	870	1186	1443	3315			
13	939	1276	1550	3570			
14	1012	1370	1661	3825			
15	1099	1479	1784	4080			
16	1180	1584	1901	4335			
17	1253	1679	2010	4590			
18	1339	1786	2134	4845			
19	1419	1887	2256	5100			
20	1500	1988	2381	5355			
21	1590	2100	2516	5610			
22	1683	2218	2654	5865			
23	1770	2332	2787	6120			
24	1874	2462	2944	6375	v		

View OpenCL Program and OpenGL Shader Source Code

To view an OpenCL program or OpenGL shader's source code, double-click it in the CodeXL Explorer. If you are using *.cl files for OpenCL program source, adding the directory as the kernel source files folder in the Debug Settings dialog lets CodeXL to identify the source. This means you can set breakpoints inside the program before it was created and built at runtime, and you can persist breakpoints between debug sessions. Source viewing, of course, only applies to objects compiled at run time, and does not work with objects created with binaries. Doubleclicking a kernel highlights the kernel function inside the program source. Applications that load program or shader source from strings or generate it at run time, have the sources saved to temporary files. Use the "File > Save As..." option to keep these files for comparison or analysis.

```
1
                                      CL Context 20penCL Program 1
7
      Redistributions in binary form must reproduce the above copyright notice, this
8
     other materials provided with the distribution.
9
10
    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
     WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIN
11
     DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INC
12
     OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY T
13
     NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EN
14
              ***********
15
16
17
18
    19
    //! Simple kernel to modify vertex positions in sine wave pattern
20
    //! @param data data in global memory
21
    22
    kernel
23
    void sineWave(
24
        __global float4 * pos,
25
       unsigned int width,
26
       unsigned int height,
27
       float time)
28
   {
        unsigned int x = get_global_id(0);
29
30
        unsigned int y = get_global_id(1);
31
        // calculate uv coordinates
32
33
       float u = x / (float) width;
34
        float v = y / (float) height;
        u = u^{*2.0f} - 1.0f;
35
        v = v^{*}2.0f - 1.0f;
36
37
38
        // calculate simple sine wave pattern
39
        float freq = 4.0f;
        float w = sin(u*freq + time) * cos(v*freq + time) * 0.5f;
40
41
        // write output vertex
42
43
        pos[y*width+x] = (float4)(u, w, v, 1.0f);
44
   }
45
```

View OpenGL Program Pipeline and Sampler Objects

To view an OpenGL Program Pipeline or Sampler object, click on the object's node in the CodeXL Explorer. This will display the object's current state in the CodeXL Properties view. The state of Program Pipelines may contain references to OpenGL Shaders. These references will be displayed in the CodeXL Properties view as links. To view an OpenGL Shader's source code, just click on the link. Here is a snapshot of the CodeXL Properties view after clicking on a Program Pipeline object.

Program Pipeline 1	
Active Program	No Program
Vertex Shader	Program 4 - Vertex Shader 1
Tess Control Shader	
Tess Evaluation Shader	
Geometry Shader	Program 4 - Geometry Shader 2
Fragment Shader	Program 5 - Fragment Shader 3
Compute Shader	
Pipeline Bound	Yes

Here is a snapshot of the CodeXL Properties view displaying the state of an OpenGL Sampler object. Note that the "Bound to the following Texture Units" field may contain more than one texture (in a comma separated list between the curly brackets).

OpenGL Sampler 1	
Bound to the following Texture Units	{0}
RGBA Red Border Color	0.000000
RGBA Green Border Color	0.000000
RGBA Blue Border Color	0.000000
RGBA Alpha Border Color	0.000000
Comparison Function	GL_LEQUAL
Comparison Mode	GL_NONE
Texture Level of Detail Bias	0.000000
Max Level of Detail	1000.000000
Min Level of Detail	-1000.000000
Magnification Filter	GL_LINEAR
Minification Filter	GL_NEAREST_MIPMAP_LINEAR
Texcoord S Wrap Mode	GL_REPEAT
Texcoord T Wrap Mode	GL_REPEAT
Texcoord R Wrap Mode	GL_REPEAT

Work with API Function Breakpoints

Any CodeXL-supported OpenCL or OpenGL function can be set as a breakpoint in the Breakpoints dialog (Alt+Shift+B). The API Functions list contains all the supported functions. The Active Breakpoints list contains all the currently selected breakpoints. Each selected breakpoint has a checkbox next to it, representing whether the breakpoint is currently enabled. Disabling breakpoints instead of removing them saves the time of finding the function each time you want to enable the breakpoint. To add a function to the breakpoints list, select it in the API functions list and press Add or double-click it. To remove a breakpoint, select it in the breakpoints list and press Remove or double click it. Using breakpoints can help you see the exact API status before the function is called, also allowing a comparison of the status before and after the function call (by pressing the Step Over button (F10) after the breakpoint hit). You also can use breakpoints to find where in your code each function is called, by looking at the Call Stack view after the breakpoint is hit. if you have debug information for the stack item, you can double-click it to see the exact location in the MDI source view.

Break point Dialog

© 2015 Advanced Micro Devices. All rights reserved.

CodeXL User Guide



Help > Tutorials > GPU Debugger Tutorial > Kernel Debugging

Kernel Debugging

PI Functions	Kernel Functions	Error / Warning	Active Breakpoints	
	its		Breakpoint Name ClEnqueueNDRangeKernel ClFlush wglSwapBuffers glDrawElements clCreateProgramWithSource clWaitForEvents Type kernel function name << Remove	Type ci APi gi APi gi APi ci APi ci APi
d wait			Remove All	

View Statistical Information About API Usage

Open the CodeXL Statistics view. The best way to overview your application's API usage is to collect the statistics for a few complete frames. Let your application run for a few frames, then suspend it by pressing Break All (F6). If you are using frame terminators, press the Step Out button (Shift+F11) to advance to the end of the current frame. Alternatively, set a breakpoint on a frame terminator, and Continue (F5) past it a few times to collect enough information. This gives you the clearest statistical information about your API usage. To change between the statistics for different OpenCL and OpenGL contexts, choose them in the CodeXL Explorer.

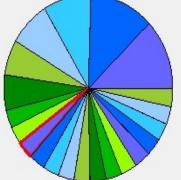
The Function Types Statistics tab shows your OpenCL and OpenGL usage by function types. The percentages here do not always add up to 100% because an API function can belong to many or none of the categories. In the Function Calls Statistics tab, you can see details of specific functions, including the function types each function belongs to. In the Vertex Batch Statistics tab, you can see details of your application's vertex batches by size, giving an overview of the vertex sizes and occurrence, as well as the total number of vertices drawn with each batch size.

% of Calls 0.00 ctions 0.00 9.22 0.00	0 0	% of Calls (, 0.00 0.00	# of Calls (Avg. per Frame) 0	,
ctions 0.00 9.22	0		0	
9.22	°	0.00		
	1 500		0	- 1
0.00	1,582	9.38	3	
0.00	0	0.00	0	
0.00	0	0.00	0	
24.55	4,211	25.00	8	
21.51	3,690	21.88	7	
Fun 0.00	0	0.00	0	
0.00	0	0.00	0	
21.46	3,682	21.88	7	
ns 55.21	9,471	56.25	18	
ons 42.97	7,371	43.75	14	
7.90	1,356	6.25	2	
	17,155		32	
Total				
			Program and Shader Functions	
	24.55 21.51 r Fun 0.00 21.46 ns 55.21 ons 42.97 7.90	24.55 4,211 21.51 3,690 r Fun 0.00 0 21.46 3,682 ns 55.21 9,471 ons 42.97 7,371 7.90 1,356 17,155	24.55 4,211 25.00 21.51 3,690 21.88 r Fun 0.00 0 0.00 21.46 3,682 21.88 ns 55.21 9,471 56.25 ons 42.97 7,371 43.75 7.90 1,356 6.25 17,155	24.55 4,211 25.00 8 21.51 3,690 21.88 7 r Fun 0.00 0 0.00 0 0.00 0 0.00 0 0 21.46 3,682 21.88 7 ns 55.21 9,471 56.25 18 ons 42.97 7,371 43.75 14 7.90 1,356 6.25 2 17,155 32 17,155 32 14

x

Jul Jul

ory Vertex Batches	Deprecated Function	ion Calls	Function Types Function
	# of Calls Function t	%	Function Name
	2,104 Texture D	12.26	glTexCoord2f
	2,104 Draw Dep	12.26	glVertex3f
Deprecated	1,579 State Chan	9.20	glMatrixMode
	1,351 Get	7.88	⚠ wglGetProcAddress
	1,053 Matrix De	6.14	glLoadIdentity
	1,053 State Chan	6.14	glBindBuffer - GL
	529 Texture	3.08	glBindTexture - GL
	528 State Chan	3.08	glDisable - GL_DEP
	527 Draw Dep	3.07	🛕 glBegin - GL_QUADS
	527 State Chan	3.07	glDisable - GL_LIG
	527 State Chan	3.07	glDisable - GL_TEX
	527 State Chan	3.07	glEnable - GL_TEX
	527 State Chan	3.07	glViewport
	527 State Chan	3.07	glBindBuffer - GL
	526 Draw	2.07	walGuanDuffor



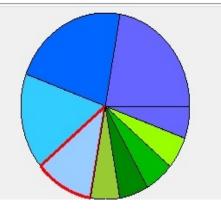
glBegin - GL_QUADS

Immediate mode rendering (glBegin-glEnd blocks) is considered an inefficient way to push primitive data into the graphic system. This mode requires making a large amount of function calls for describing the primitives' data.

x

Instead, consider using VBOs (Vertex Buffer Objects) or Vertex Arrays, which enable pushing large data chunks at once.

Function Types	Function Calls	Deprecated Function	Calls History	Vertex Bat	tches	
Function Name	Deprec	ation Reason	# of Calls	% of Calls	Deprecated at Ver	Removed at Version
\land glTexCoord2f	Comple	etely Deprecated Functi	2,104	12.26	OpenGL 3.0	OpenGL 3.1
\Lambda glVertex3f	Comple	etely Deprecated Functi	2,104	12.26	OpenGL 3.0	OpenGL 3.1
🛕 glMatrixMod	e Comple	etely Deprecated Functi	1,579	9.20	OpenGL 3.0	OpenGL 3.1
🋕 glLoadldentit	y Compl	etely Deprecated Functi	1,053	6.14	OpenGL 3.0	OpenGL 3.1
Å glBegin 🛛 🕅	oadIdentity pl	etely Deprecated Functi	526	3.07	OpenGL 3.0	OpenGL 3.1
\Lambda glEnd	Comple	etely Deprecated Functi	526	3.07	OpenGL 3.0	OpenGL 3.1
Å glOrtho	Comple	etely Deprecated Functi	526	3.07	OpenGL 3.0	OpenGL 3.1
🛕 glPopMatrix	Comple	etely Deprecated Functi	526	3.07	OpenGL 3.0	OpenGL 3.1
🛕 glPushMatrix	Comple	etely Deprecated Functi	526	3.07	OpenGL 3.0	OpenGL 3.1
🛕 glMultMatrix	d Comple	etely Deprecated Functi	1	0.01	OpenGL 3.0	OpenGL 3.1
Total (10 iten	ns)		9,471	55.21		



<u>/</u> !	glLoadIdentity
Dep	recated Feature: Fixed Pipeline Vertex Processing
Oper	function glLoadIdentity was marked as deprecated in nGL version OpenGL 3.0 and was removed from OpenGL at ion OpenGL 3.1.
	It is recommended to avoid using deprecated functions, as their patibility with future OpenGL versions is not assured.

x

۸

¥

Function Types	Function Calls	Deprecated Function	Calls History	Vertex Batch	es	
/ertices per Batc	h	# of Batches	% of B	atches	# of Vertices	% of Vertices
1 1		16,368	26.67		16,368	0.03
0 6		123	0.20		738	0.00
1 24		8,061	13.13		193,464	0.38
12 🚺		4,092	6.67	1	294,624	0.58
1,536		32,736	53.33		50,282,496	99.01
otal		61,380	100%		50,787,690	100%
8000				72 Vertex	Batch	
8000						with 6.67% of your
8000 4000 0000			You o	iraw 0.5801%		with 6.67% of your
8000 4000 0000 6000			batc	lraw 0.5801% hes	of your vertices	
22000 28000 24000 16000 12000 8000 4000			batcl	fraw 0.5801% hes t is good practic	o of your vertices e to draw as many v	with 6.67% of your vertices using as few batche the application's rendering

Kernel Debugging

On AMD GPU hardware ,CodeXL allows debugging OpenCL kernels in real time on AMD GPU hardware, with only a single machine and single device, and with hardware results.

Enter Kernel Debugging

There are three ways to enter kernel debugging:

- After starting debugging and suspending the process (with Break / F6 or an API function breakpoint), use the Step command or an API function breakpoint to get to an API call of clEnqueueNDRangeKernel or clEnqueueTask. Then, press Step Into (F11) to start debugging the kernel.
- Add the kernel function name as a kernel function

breakpoint in the breakpoints dialog (Alt+Shift+B), by switching to the Kernel Functions tab or by manually typing in the kernel name in the Active Breakpoints list. When a kernel matching the function name starts executing, the debugged process stops at the kernel's beginning.

From the CodeXL Explorer view, open the OpenCL program, and set a breakpoint inside the code. The application stops inside the kernel when it hits the breakpoint.

Kernel Debugging Control

While in kernel debugging, press step over (F10) to step to the next line in the same scope, step in (F11) to enter functions called inside the kernel, or step out (Shift+F11) to leave the current function or kernel.

You can also set kernel source breakpoints by selecting a line in the kernel and pressing Add Breakpoint (F9), or clicking in the margin next to the line. Press Continue (F5) to run up to the breakpoint. You can also set a breakpoint inside a function to enter that function (as with stepping in).

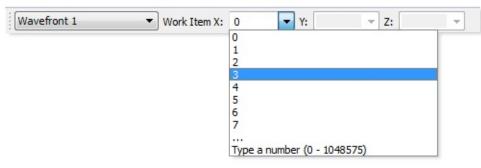
Focus On a Specific Work-Item

CodeXL's Current Work-Item toolbar lets you select a single work-item to focus on in CodeXL. If the toolbar is not open, right-click in the toolbars area, and select it from the dropdown menu. The combo-boxes for X, Y, and Z coordinates are enabled and offer values depending on the currently debugged kernel's N-Dimensional global work size. Selecting a work-item has the following effects:

- The locals and Watch views display variable values for this work-item.
- When stepping through kernel debugging, "if" or "else"

clauses not entered by the selected work-item are skipped.

The work-item is selected in the MultiWatch views.



Inspect Variable Values

Use the Local and Watch views to inspect variable values. The Locals view displays all variables that are currently in-scope. Use the Watch view to dereference pointers (*p) or access array members (a[150]).

The values shown in all these views are the at the current work-item, as selected by the current work-item toolbar.

Compare Variable Values Across Work-Items and Work-Groups

Use CodeXL Multi-Watch views to compare the values of variables across various work-items and work-groups. To open a Multi-Watch view, select it from the Views toolbar or menu, or right-click a variable and select "OpenCL Multi-Watch...". The view's variable selection combo-box offers all variables that can be viewed (similar to the Locals view list). Select a variable to view its values. The main view displays a spreadsheet or graphical representation of the values along a sliding scale ranging from the lowest value to the highest. Move the pointers in the Active Range slider to quickly highlight values outside a given range.

In Data (spreadsheet) view, the variable values are placed into cells according to their work-item coordinates. Work-groups

are alternately colored white and gray, letting you easily see the edge of a work-group.

In Image view, variables inside the range are colored on a gradient scale from black (lowest) to white (highest). Values below the range are red; values above the range are purple; work-items that are out of scope are yellow.

For three-dimensional work sizes, use the Z coordinate slider to navigate between the various XY planes in the global work.

The "hovered" controls display the location, value, and color for the mouse cursor's location. Click the view to set the current values to the "selected" control for an easy comparison against a single work-item. This also centers the Data view on the selected item.

OpenCL Mult	ti-Watch 3					x
Image view	Data view				Multiple Kernel V	Vork Item Watch
		sum.s2			Variable Name:	sum.s2 V
					Variable Type:	N/A
	C SNM	MALL .	1 Martin		Global Work Offset	: N/A
11/2			Mater 1		Global Work Size:	N/A
	- Jen		572		Local Work Size:	N/A
	6	S. Carto	10 pt to the		Hovered W. Item:	X: 952, Y: 84
1			When is ?		Selected W. Item:	N/A
		Silver.			Hovered Value:	798
		Contraction of the	the state		Selected Value:	N/A
		A	141- 141 172-10		Hovered Color:	R: 199, G: 199, B: 199, A: 255
					Selected Color:	N/A
		SA !	Later and		Place the mouse po	inter over the image pixel
		A	n		to view the texel in	formation
		UL N	1		Adjust variable valu	e active range:
	10	10012	T AV		0	1020
			Antime		-	
					0 min	max 1020
				and the second		
				A CARL		
	N3		NA.			
			U	and a state of the		
<				>		

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > GPU Debugger Tutorial > Keyboard Shortcuts

Keyboard Shortcuts

Views and Dialogs				
New Project wizard	Ctrl+N			
Debug Settings dialog	Ctrl+O			
Breakpoints dialog	Alt+Shift+B			
Common Debugging				
Go (Start / Continue Debugging)	F5			
Stop Debugging	Shift+F5			
API-Level Debugging				
Break at next API function call	F6			
Step to next API function call	F10			
Step into kernel debugging	F11			
Step to next frame terminator	Ctrl+F11			
Kernel Debugging				
Step Over	F10			
Step Into function	F11			
Step Out of function / kernel	Shift+F11			
Set / remove breakpoint at kernel source	F9			

 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial

CPU Profiler Tutorial

The CPU Profiler component in CodeXL is used for performance analysis and tuning of applications running on CPU. It is used to identify the various factors affecting the performance of the profiled application or the entire system.

This tutorial provides step-by-step instructions for using CodeXL to analyze the performance of an application program. It consists of the following modules.

Preparing an Application for CPU ProfilingCreate a CodeXL Project for CPU ProfilingSet the global options for CPU ProfilingSet the CPU Profile Session OptionsAnalysis with Time-based Profiling(TBP)Analysis with Event-based Profiling(EBP)Analysis with Instruction Based Sampling(IBS)Analysis with Cache Line Utilization (CLU)Settings for C++ Inline FunctionsSettings for CPU Profiler to run on Virtual MachineHow to use CPU Profiler Control APIs

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Preparing an Application for CPU Profiling

Preparing an Application for CPU Profiling

The CodeXL CPU Profiler uses the debug information generated by the compiler to show the correct function names in various analysis views and to attribute the collected samples to source statements in Source View. Otherwise, the results of the CPU Profiler would be less descriptive, displaying only the assembly code, and the functions would be reported as **NO SYMBOL**.

Generate Debug Information on Windows

When compiling an application in release mode, you can produce the debug information to enable CodeXL to perform an analysis. When using Microsoft Visual C++ to compile the application in release mode, set the following options to ensure that the debug information is generated and saved in a program database file(with a .pdb extension).

To set the compiler option to generate the debug information for a Win32 application in release mode:

- 1. Right click on the project and select **Properties** menu item.
- 2. In the **Configuration** list, select **Active(Release)**.
- 3. In the **Platform** list, select **Active(Win32)**.
- 4. In the project pane, expand the Configuration Properties folder, then expand the C/C++ folder.
- 5. Select **General**.
- In the work pane, select Debug Information Format, and from the drop-down list select Program Database (Zi) or Program Database for Edit & Continue (/ZI).

AMDTClassicMatMul Property Pages			? ×
Configuration: Release	V Platform: Active(Win32)	~	Configuration Manager
 ✓ Configuration Properties General Debugging VC++ Directories ✓ C/C++ General Optimization Preprocessor Code Generation Language Precompiled Headers Output Files Browse Information Advanced All Options Command Line > Linker > Manifest Tool > XML Document Generator > Browse Information > Build Events > Custom Build Step > Code Analysis 	Additional Include Directories Additional #using Directories Debug Information Format Common Language RunTime Support Consume Windows Runtime Extension Suppress Startup Banner Warning Level Treat Warnings As Errors Warning Version SDL checks Multi-processor Compilation	Program Database (/Zi) Yes (/nologo) Level3 (/W3) No (/WX-)	
Debug Information Format Specifies the type of debugging information generated by the compiler. This requires compatible linker set (/Z7, /Zi, /ZI)			
		ОК	Cancel Apply

- 7. In the project pane, expand the Linker folder; then expand the **Debugging** folder.
- 8. In the Generate Debug Info list, select Yes (/DEBUG).

AMDTClassicMatMul Property Pages			? X
Configuration: Release	✓ Platform: Active(Win32)	~	Configuration Manager
Language A Precompiled Heade Output Files Browse Information Advanced All Options Command Line Linker General Input Manifest File Debugging System Optimization Embedded IDL Windows Metadata Advanced All Options Command Line Manifest Tool Manifest Tool XML Document Genera Browse Information Build Events Custom Build Step Code Analysis	Generate Debug Info Generate Program Database File Generate Full Program Database File Strip Private Symbols Generate Map File Map File Name Map Exports Debuggable Assembly	true \$(OutDir)\$(TargetName).pdb No No No No Solution No No No Solution Solution No No No Solution <	
< >		ОК	Cancel Apply

Generate Debug Information on Linux

The application must be compiled with the -g option to enable the compiler to generate debug information. Modify either the Makefile or the respective build scripts accordingly.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Create a CodeXL Project for CPU Profiling

Create a CodeXL Project for CPU Profiling

To start profiling their application, you must create a new CodeXL Project and set the appropriate settings. The following steps explain how to create a new CodeXL Project.

- Once CodeXL is launched, click on Create New Project link in the main pane, or select File > New Project... The Create a new CodeXL Project dialog box is displayed.
- 2. Select the **General** tree node on the left pane.
- 3. In the **Executable Path** box, click **Browse**, and select the application to be profiled.
- 4. By default, CodeXL uses the name of the application as the project name. To modify the project name, edit the **CodeXL Project Name** box.
- 5. Edit the other boxes that are required for executing the application to be profiled.
 - Working Directory
 - **Command Line Arguments** (optional)
 - Environment Variables (optional)
 - Source Files Directories (optional)

🗷 Create a new CodeXL Project		Х
General Debug	General	
 Profile CPU Profile Application Timeline Trace GPU Profile: Perf. Counters 	CodeXL Project Name: Target Host: Remote Host Address:	AMDTClassicMatMul
Power Profile	Target Application	
Frame Analysis Y Analyze	Application Type:	Desktop Application O Windows Store App
> Kernel/Shader Build Options	Executable Path:	C: \Users \Public \Documents \CodeXL \Examples \ClassicMatMul \Release \AMDTClassicMatMul.exe
HLSL Build Options	Working Directory:	C: \Users \Public \Documents \CodeXL \Examples \ClassicMatMul \Release
	Command Line Arguments:	
	Environment Variables:	m
	Source Files	
Restore Default Settings	Source Files Directories: Source Files Resolution	
	Root Directory:	
Restore Default Settings		OK Cancel

6. Click **OK**.

A new CodeXL project is created and displayed in the Session Explorer window.

 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Create a CodeXL Project for CPU Profiling > Creating CPU Profiling Project for Windows Store (Metro) Apps

Creating CPU Profiling Project for Windows Store (Metro) Apps

- Once CodeXL is launched, click on Create New Project link in the main pane, or select File > New Project... The Create a new CodeXL Project dialog box is displayed as shown in the previous screenshot.
- 2. Select the **General** tree node on the left pane.
- 3. Select "**Application Type**" as "**Windows Store App**". Please note that Store Apps are only available on Windows 8.x or later versions.
- 4. In the **Executable Path** box, click **Browse**. The **Select Windows Store app** dialog will be displayed. Select the Store App to be profiled. Select **OK**.

XL	Select Windows Store app	×
Č	Alarms	^
	Calculator	
	Calendar	
0	Camera	
1	Finance	
1	🕇 Food & Drink	
Ľ	Games	~
Us	er Model ID: pwsCalculator_8wekyb3d8bbwe!/	Арр
	OK Cancel	

- 1. Configure other settings as described in the previous section item 4, 5.
- 2. Click **OK**.

A new CodeXL project for Windows Store App is created and displayed in the Session Explorer window.





Help > Tutorials > CPU Profiler Tutorial > Set the global options for CPU Profiling

Set the global options for CPU Profiling

Specify the global CPU profile settings by modifying the **CodeXL Options** dialog box. The following steps explain how to do this.

- Select the profile mode by selecting Profile > Profile
 Mode CPU: Time-based Sampling or by clicking the Profile icon I
- Select Tools > Options... menu item. The CodeXL Options dialog box is displayed.
- 3. Select the **General** tab, as shown in the following screenshot.

General	GPU Debug	CPU Profile	Analyze			
	GPO Debug	CFOFIOIIE	Analyze			
Logging						
Debug lo	g level:				DE	EBUG 🔻
Log Files	and Images Dir	ectory				
C:\User	s\salgrana\Appl	Data\Local\Temp)			
Connect	ion					
	using an HTTP p					
Server:	www.myproxy	y.example			Port	8080 🗘
Remote	Debugging Ports	s:		2023 🗘 2024	2025 🖨	2026 🗘
Advance	d					
	-					
Floating	point precision	(maximum numb	er of significant d	igits):		8 🜩
Source F	iles					
Alert	when a source	file is missing				

- 4. Select **Alert when a source file is missing** check box if you want CodeXL to pop up an alert message dialog box if source files are missing.
- 5. Select the **CPU Profile** tab, as shown in the screenshot in the below sub-section 'Setting Symbol Server Path'.

 $[\]ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Set the global options for CPU Profiling > Setting Symbol Server Path

Setting Symbol Server Path

- 1. If any additional debug symbol paths are to be specified to find the required .pdb files, select the **Additional debug symbol paths** check box, **browse**, and select the folder in which PDB files are kept.
- 2. Select **Symbol Server Directories** to enable the support of Microsoft symbol server technology. Set the download directory and select Microsoft symbol server link. Additional symbol server paths can be added to this list. (See the help file for a detailed description of the available options.)

General	GPU Debug	CPU Profile	Analyze			
		<u></u>	- and yet			
Symbols	;					
🗹 Add	litional debug sy	mbol paths	C:\LocalSymbols;			
✓ Sy	mbol Server Dir	ectories				
Down	load directory	C:\LocalSymbo	ols:			
			5157			
	New	Remove		Move Up	Move <u>D</u> own	
	http://msdl.mi	crosoft.com/	'download/symbols			
	http://msdl.mi	crosoft.com/	'download/symbols			
	http://msdl.mi	crosoft.com/	'download/symbols			
	http://msdl.mi	crosoft.com/	'download/symbols			

3. Select the block size for disassembly fetch while navigating through disassembly code. The default size of 1024 is usually sufficient.

CodeXL User Guide



Help > Tutorials > CPU Profiler Tutorial > Set the General Profile Session Options

Set the General Profile Session Options

- 1. Click the **Profile > Profile Settings** menu item.
- 2. Click the **Profile** tree node in the left pane.
- 3. Select one of the CPU profile types in the profile type combo box. The corresponding description will appear.
- 4. Select the profile scope:
 - **Single Application Profile:** when this option is selected, CodeXL will launch the profiled application (classic.exe in this case), and will collect profile data only from this process.
 - **System-Wide Profile:** when this option is selected, CodeXL will not launch the profiled application, and will collect profile data from all the currently running processes.

System-Wide Profile with focus on application: when this option is selected, CodeXL will launch the profiled application (classic.exe). It will collect profiled events data both from classic.exe and other running processes on the machine. For classic.exe, CodeXL will also collect call stack information.

🔟 CodeXL Project Settings		X
General Debug V Profile V CPU Profile	ille Type e Session Type: CPU: Time-based Sampling Identify the hot spots in a program that are consuming the most time e Scope: Single Application Profile (AMDTClassidMatMul.exe) System-Wide Profile System-Wide Profile with focus on application (AMDTClassidMatMul.exe) 	X
Restore Default Settings		OK Cancel





Help > Tutorials > CPU Profiler Tutorial > Set the CPU Profile Session Options

Set the CPU Profile Session Options

The CPU Profile view is used to control the profile data collection options. The selected settings persist and apply to future profile data collection sessions that are started within a project; they persist until the profile settings are modified.

- 1. Click the **Profile > Profile Settings** menu item.
- Click the **Profile** tree node in the left pane. The **CPU Profile** settings page is displayed (see the following screenshot).
- 3. Select **Collect call stack details** to enable the collection of call stack data to analyze caller-called relationships and call frequency.

Note: The collection of call stack is not supported for Java or CLR based applications. For those applications, this checkbox would be unchecked and disabled.

Set the value for **Collect for code executed in** drop box, to apply call stacks collection only to code sampled in User space, Kernel space or both (User space and Kernel space).

Set the value for **Collect Call Stack every** spin box, to specify how often a CSS sample is taken. If the selected value is N for example, then a CSS sample is taken every N regular samples. By default it is set to 1.

Set the value for **Call Stack Collection Depth**. Use Maximal for high accuracy, and Minimal for lower performance overhead.

Note: This option is set separately for Time-Based Sampling and other CPU Profiling session types.

Check the **Reproduce missing call stack info** check box, to perform additional analysis to overcome framepointer omission (FPO) in 32-bit apps and lack of unwind info in 64-bit. The profiler will store additional data during the profile session and require more time during post-session processing.

Note: This option is set separately for Time-Based Sampling and other CPU Profiling session types.

- 4. In the **Data Collection Schedule** section:
 - Select **Throughout entire duration** if you want the data to be collected throughout the entire execution of the profile session.
 - Select **Start profile with data collection paused** if you want to start the application and then decide when data collection should be started.
 - Select **Scheduled:** if you want to set the data collection start after and end after values. Once this option is checked, set the values for the start / end timing of the data collection.
- 5. Select / Unselect cores in the cores tree under **Profile Hardware Scope**, when you want to limit the execution of the program to the selected cores in a multicore system.

XL	CodeXL Project Settings			Х
~	General Debug Profile V CPU Profile	Call Stack Collection	Collect call stack details for current session type - CPU: Time-based Sampling	
	Custom Application Timeline Trace GPU Profile: Perf. Counters	Collect for code executed in: Collect call stack every:	User mode	
~	Power Profile Frame Analysis Analyze > Kernel/Shader Build Options	Call stack collection depth Time-Based Sampling: Other CPU Profiling session types:	High (128 levels) ▼ Reproduce missing call stack info Low (32 levels) ▼ Reproduce missing call stack info	
	HLSL Build Options	Data Collection Schedule		
		Collection Schedule:	Throughout entire duration	
			○ Start profile with data collection paused	
			◯ Scheduled:	
			Start data collection after 0 🜩 seconds	
			End data collection after 0 🔹 additional seconds	
			Then, terminate the process	
		Profile Hardware Scope		
		 ✓ CPU 0 ✓ Core 0 ✓ Core 1 ✓ Core 2 ✓ Core 3 ✓ Core 4 ✓ Core 5 ✓ Core 6 ✓ Core 7 	: Dxff	
Re	store Default Settings		OK Cancel	

 $\ensuremath{\textcircled{\sc c}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Time-based Profiling (TBP)

Analysis with Time-based Profiling (TBP)

Time-based profiling (TBP) identifies the hot spots in a program that are consuming the most time. Hot spots are good candidates for further investigation and optimization.

<u>Collecting TBP profile Data</u> <u>Changing the TBP profile collection options</u> <u>Analyzing the TBP profile data</u>



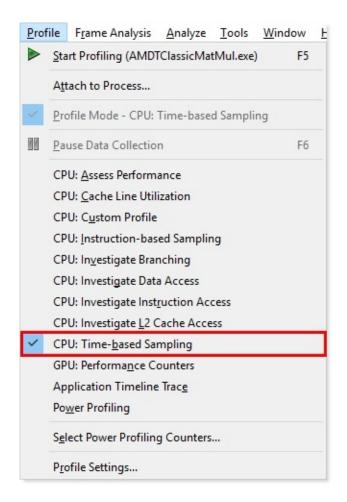


Help > Tutorials > CPU Profiler Tutorial > Analysis with Time-based Profiling (TBP) > Collecting TBP profile Data

Collecting TBP profile Data

To collect the TBP data, do the following.

- 1. <u>Create a new CodeXL Project</u>.
- 2. In the toolbar, select Time-based profile in the profile configuration list, as shown in the following screenshot.



3. Set the profile session options as explained in <u>Set the</u> <u>CPU Profile Session settings</u>.

4. Click the Start button in the toolbar or select **Profile > Start Profiling,** as shown below.

<u>P</u> rof	ile F <u>r</u> ame Analysis <u>A</u> nalyze <u>T</u> ools <u>W</u> indow	
	Start Profiling (AMDTClassicMatMul.exe) F5	
	Attach to Process	
~	Profile Mode - CPU: Time-based Sampling	
88	Pause Data Collection F6	
	CPU: Assess Performance	
	CPU: Cache Line Utilization	
	CPU: Custom Profile	
	CPU: Instruction-based Sampling	
	CPU: Investigate Branching	
	CPU: Investigate Data Access	
	CPU: Investigate Instruction Access	
	CPU: Investigate L2 Cache Access	
~	CPU: Time- <u>b</u> ased Sampling	
	GPU: Performance Counters	
	Application Timeline Trace	
	Po <u>w</u> er Profiling	
	Select Power Profiling Counters	
	P <u>r</u> ofile Settings	

CodeXL begins data collection and launches the specified application program. The start icon is disabled until profiling is in progress. The console window is open, and the launched application is running in the console window. The status bar shows the profile duration while profiling is running; this is the percentage completion of data translation during data translation phase.

AMDTClassicMatMul (Running) - CodeXL Profile	e Mode (CPU: Time-based Sampling)		– 🗆 X
<u>File Edit View Debug Profile</u> F <u>r</u> ame Analys	sis <u>A</u> nalyze <u>T</u> ools <u>W</u> indow <u>H</u> elp		
🕅 🕤 🐨 🔟 🖉 🕨 Local 🔻 🛄 🔳			
CodeXL Explorer 🗗 🗙	Apr-07-2016_15-50-43 (CPU: Time-based Sampling)		
	Profile Overview		
🗸 췕 AMDTClassicMatMul Profile Mode (CPU:			
✓ ♂ CPU: Time-based Sampling	5 Hottest Functions	Display: System Modules Hi	dden Hotspot Indicator Timer 🔻 🔨
✓ ₫ Apr-07-2016_15-50-43	Function Samples % of Hotspot Sam	aler Medule	
		pies Middule	
Mod C:\ProgramData\AMD\C Call	CodeXL\Examples\ClassicMatMul\Release\AMDTClassicMatMul.exe		- 🗆 X
i Fun≪Matrix multiplicatio	n samle		^
Initializing matrice			
Multiplying matrices			
Invoke inefficient i	mplementation of matrix multiplication		
			>
			^
Properties			
CPU: Profile Session			
Profile Type Time			
Session Name Apr-			
Executable Path: C:\Progr			
\Examples \ClassicMatMul \ \AMDTClassicMatMul.exe			
Working Directory: C:\Pro			
Examples ClassicMatMull			v
Profile Scope: Single Appli			✓ ncti
Profile Start Time Apr-07-2016_15-50-43			
Profile duration: 5 seconds			0%





Help > Tutorials > CPU Profiler Tutorial > Analysis with Time-based Profiling (TBP) > Changing the TBP profile collection options

Changing the TBP profile collection options

CodeXL lets you modify the default profile collection settings. To do this:

- 1. Set the profile session options as explained in <u>Set the</u> <u>CPU Profile Session settings</u>.
- 2. By default, the sampling interval for TBP is 1 millisecond. To modify the sampling interval, click **Profile > Profile Settings** and click on the **CPU Profile > Custom** node in the left pane tree in the **Edit CodeXL Project Settings** dialog box. The **Custom CPU Profile Monitored Events** page will be displayed.
- 3. Click the **Interval** field and set the required sampling interval.
- 4. Click the **OK** button to save the modified profile settings.

🗷 Create a new CodeXL Project					×
Create a new CodeXL Project General Debug Profile CUU Profile CUStom Application Timeline Trace GPU Profile: Perf. Counters Power Profile Frame Analysis Analyze Nernel/Shader Build Options HLSL Build Options	Custom CPU Profile Monitored Events Add an Event to the 'Monitored Events' list to include Available Events > Assess Performance > Cache Line Utilization > Instruction-based Sampling > Investigate Branching > Investigate Data Access > Investigate Instruction Access > Investigate L2 Cache Access > Time-based Sampling [E100] Timer Event > Events by Hardware Source > All Events	rofile session Monitored Events Name [E100] Timer Event	Unit Mask	User Mode	X Kernel Mode
	[E100] Timer Event Samples are taken at discrete time intervals.	ζ			
Restore Default Settings				OK	Cancel





Help > Tutorials > CPU Profiler Tutorial > Analysis with Time-based Profiling (TBP) > Analyzing the TBP profile data

Analyzing the TBP profile data

This page explains how to analyze and navigate to the various profile views provided by the CodeXL CPU Profiler. It consists of the following modules.

Overview page Modules view Functions view Call Graph view Source Code view





Help > Tutorials > CPU Profiler Tutorial > Analysis with Time-based Profiling (TBP) > Analyzing the TBP profile data > Overview page

Overview page

When data collection is complete, CodeXL processes the data and displays the results.

Initially the **Profile Overview** is displayed with the following profile data.

- 5 Hottest Functions table
- 5 Hottest Modules table
- **Execution** section
- **Profile Details** section

AMDTClassicMatMul - CodeXL Profile N	Ande (CPU)	Time-hased (samilion)	- D X
Eile Edit View Debug Profile Fran	ne Analysis	Analyze Tools Millidow Help	
CodeXL Explorer			2016_15-53-25 (CPU: Time-based Sampling) 🔲 🔳
	1.00	D Profile Overview	
 AMDTClassicMatMul Profile Mode (O CPU: Time-based Sampling O Apr-07-2016_15-50-43 		5 Hottest Functions Display: System Mo	odules Hidden Hotspot Indicator Timer 🗸
Overview		Function Samples % of Hotspot Samples Module	
Modules		classic_multiply_matrices(void) 502 98.62% AMDTClassicMatMul.exe	
Call Graph		initialize_matrices(void) 6 1.18% AMDTClassicMatMul.exe BHDrvx64.sysl0xf800ff8ac2b3 1 0.20% BHDrvx64.sys	
 ✓ Ø Apr-07-2016_15-50-56 Ø Overview 	5	Hottest Processes 5 Hottest Modules	
 Modules Call Graph Functions G Apr-07-2016_15-53-25 O Overview Modules Call Graph Functions 		Process PID Samples % of Hotspot Samples Module AMDTClassicMatMul.exe 9964 542 92.33% Image: Conhost.exe 8860 45 7.67% K K K K K K K K Execution K K K K K Execution K K K K Forfile Session Directory: C: (ProgramData \AMD\CodeXL \Examples \ClassidMatMul\Release \AMDTClassidMatMul.exe K Profile Session Directory: C: (Users\salgrana \AppData\Roaming \AMD\CodeXL \AMDTClassidMatMul_ProfilerOutput\AppCommand Line Arguments: K	1 0.20%
		Environment Variables;	
CPU: Profile Session Profile Type Time-based Sampling		CPU Affinity: 0xff Profile Scope: Single Application	
Session Name Apr-07-2016_15-53-25		Call Stack Sampling: True Call Stack Information: Scope - User mode, Depth - 128, F	Frequency - per 1 samples, FPO - On
Executable Path: C: \ProgramData\AMD\CodeXL \Examples\ClassicMatMul\Release \AMDTClassicMatMul.exe		Profile Details	
Working Directory: C:\ProgramData\AMD\CodeXL \Examples\ClassicMatMul\Release		Profile Session Type: Time-based Sampling Monitored Events	v
Profile Scope: Single Application Profile Start Time Apr-07-2016_15-53-25	, [1 The 5 Hottest Functions usually indicate the most significant performance bottlenecks of your application. Select a	different Hotspot Indicator to display hottest functi
Ready			

From the overview page you can navigate to various pages to analyze the profile data.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Time-based Profiling (TBP) > Analyzing the TBP profile data > Modules view

Modules view

 To navigate to Modules view, do one of the following:

 In CodeXL Explorer pane, double-click on Modules node in the profile session tree entry created for this profile run.

The **Modules** view is displayed.

In **Overview** page, right-click on **5 Hottest Modules** table entry, and select **Display in Modules View**. The **Modules** view is displayed.

2. Click on the **Display: All Data, All Modules** link to set the Display Settings.

The **Display Settings** dialog box is displayed to let you set the various display settings.

AMDTClassicMatMul - CodeXL Profile Mo			□ ×
🏷 🚫 🕶 🙋 🕨 Local 🔻 🖩 🔳			
CodeXL Explorer	₽×	50-43 (CPU: Time-based Sampling) 🗵 🛛 💆 Apr-07-2016_15-50-56 (CPU: Time-based Sampling) 🔟 🕺 Apr-07-2016_15-53-25 (CPU: Time-based Sam	npling) 🗵 🔳
	XL	Profile Overview	
 AMDTClassicMatMul Profile Mode (CF O CPU: Time-based Sampling CPU: Time-based Sampling 	PU:	5 Hottest Functions Display: System Modules Hidden Hotspot Indicator	r Timer 🔻
 Apr-07-2016_15-50-43 Overview 		Function Samples % of Hotspot Samples Module III Display Settings	
Modules		Classic multiply matrices(void) 502 98.62%	
🚊 Call Graph		initialize_matrices(void) 6 1.18% AMDTClassicMat	
§ Functions		Hermitian Strategy St	
✓ ₫ Apr-07-2016_15-50-56		Timer	
Overview		5 Hottest Processes 5 Hottest	
Modules		Process PID Samples % of Hotspot Samples Module pot Samples	;
Call GraphFunctions		AMDTClassicMatMul.exe 9964 542 92.33%	
 Functions <u> <u> </u></u>		conhost.exe 8860 45 7.67%	
 Overview 			
Modules		()	>
🖺 Call Graph			S. 1
§ Functions		Execution	^
		General Target Path: C: \ProgramData\AMD\CodeXL\Examples\ClassidMatMulRelease\AMDTClassidMatM	
		Working Directory: C: ProgramData AMD CodeXL Examples ClassicMatMul\Release	
		Profile Session Directory: C:\Users\salgrana\AppData\Roaming\AMD\CodeXL\AMDTClassicMa	
Properties	đ×	Command Line Arguments: CPU Cores	
CPU: Profile Session	٨	Environment Variables:	
Profile Type Time-based Sampling		CPU Affinity: 0xff Profile Scope: Single Application Core 0 Core 1	
Session Name Apr-07-2016_15-53-25		Call Stack Sampling: True Call Stack Information: Scope - Use 🗹 Core 2 🖓 Core 3 es, FPO - On	
Executable Path: C:\ProgramData\AMD\CodeXL Examples\ClassicMatMul\Release		☑ Core 4 ☑ Core 5 🗸	
\AMDTClassicMatMul.exe		Profile Details Display data per:	
Working Directory: C:\ProgramData\AMD\CodeXL \Examples\ClassicMatMul\Release		Profile Session Type: Time-based Sampling Monitored Events	¥
Profile Scope: Single Application Profile Start Time Apr-07-2016_15-53-25			y hottest functi
Ready	•	OK Cancel	

3. Click \mathbf{OK} button to save the Display filter settings.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Time-based Profiling (TBP) > Analyzing the TBP profile data > Functions view

Functions view

•

•

1. To navigate to the Functions view, do one of the following.

In CodeXL Explorer pane, double-click on the **Functions** node in the profile session tree entry created for this profile run. The **Functions** view is displayed with function entries from all the profiled processes and the load modules.

In **Overview** page, right-click on the **5 Hottest Modules** table entry, and select **Display in Functions View**.

The **Functions** view is displayed with function entries from the selected load module.

In **Overview** page, right-click on the **5 Hottest Functions** table entry, and select **Display in Functions View**.

The **Functions** view is displayed with function entries from all the profiled processes and the load modules.

In **Modules** view, select a Process from the **Processes** table, and select **Display in Functions View**.

This displays the functions from the selected process and all its load modules.

In **Modules** view, select a Module from the **Modules filtered by selected processes** table, and select **Display in Functions View**.

This displays the functions from the selected load module.

2. Click on **Display: All Data, All Modules** link to set the display settings.

The **Display Settings** dialog box is displayed to let you set the various display settings.

- 3. Click **OK** button to save the Display filter settings.
- 4. Click on **modules shown**, **modules hidden** link to set the module filter settings to be used in the Functions view. The **CodeXL Modules Filter** dialog box is displayed to let you set the module filter settings.

🗷 AMDTClassicMatMul - CodeXL Profile Mode	(CPU: Time-based Sampling)			– 🗆 X
<u>File Edit View D</u> ebug <u>P</u> rofile F <u>r</u> ame An	alysis <u>A</u> nalyze <u>T</u> ools <u>W</u> indow <u>H</u> elp			
🏷 🔿 🕶 🖉 🕨 Local 🔻 📗 🔳				
CodeXL Explorer B	× 👌 Apr-07-2016_15-50-43 (CPU: Time-based S	Sampling) 🛛 👌 Apr-07-2016 15-	-50-56 (CPU: Time-based Sampling) 🔝 🛛 🚺	🖞 Apr-07-2016_15-53-25 (CPU: Time-
(III)		Functions		
 AMDTClassicMatMul Profile Mode (CPU: O CPU: Time-based Sampling O Apr-07-2016_15-50-43 	Functions	Display: <u>All Data, All Modules</u>	14 modules shown, 0 modules hidden Pro	cess All Processes 🔻
OverviewModules	Function classic_multiply_matrices(void) for PfFileInfoNotify	Module CodeXL Modules Filter	Timer 🔥	×
 Call Graph Functions 	wcsstr	Modules of System-wide Profiling(2	Processes):	
 Apr-07-2016_15-50-56 Overview Modules Call Graph Functions Apr-07-2016_15-53-25 Overview Modules Call Graph Call Graph Functions 	 initialize_matrices(void) qsort_s qsort_s RtlAcquireSRWLockExclusive ExAcquirePushLockExclusiveEx GetLastError initialize_narrow_environment CalcVisRgn KeSetEvent RtlCopyMemoryNonTemporal setjmpex ExAllocatePoolWithTag FlsGetValue RtlAllocateHeap 	Image: Second	Path e C:\ProgramData\AMD\CodeXL\Exampl C:\Windows\SysWOW64\ C:\Windows\SysWOW64\ C:\Windows\System32\ C:\Windows\System32\ C:\Windows\System32\ C:\Windows\System32\ C:\Windows\System32\ C:\Windows\System32\ C:\Windows\System32\ C:\Windows\System32\ C:\Windows\System32\ C:\Windows\System32\ C:\Windows\System32\ C:\Windows\System32\	▲ les\ClassicMatMul\Release\
	 RtIDeactivateActivationContextUnsafel RtIRestoreLastWin32Error 	Display System modules in Modu		
Properties 🗗		Select all modules		
CPU: Profile Session	▲ rand			OK Cancel
Profile Type Time-based Sampling	4 cdd.dll!0xf96137f207be	caa.aii		
Session Name Apr-07-2016_15-50-43	FltlsCallbackDataDirty	FLTMGR.SYS	1	
Executable Path: C:\ProgramData\AMD\CodeXL	RtlLookupFunctionEntry	ntdll.dll	1	
\Examples\ClassicMatMul\Release \AMDTClassicMatMul.exe	RtlActivateActivationContextUnsafeFast	t ntdll.dll	1	
•	RtIDosPathNameToRelativeNtPathNameToRelativ	ne_U_WithStatus ntdll.dll	1	
Working Directory: C:\ProgramData\AMD\CodeXL \Examples\ClassicMatMul\Release	RtlReleaseSRWLockExclusive	ntdll.dll	1	~
Profile Scope: Single Application	Euroctions with a high sample count usual	v indicate performance bottlepecko. So	ort the table according to a specific metric to hi	inhight potential bottleneck functions
Profile Start Time Apr-07-2016_15-50-43		y marcate performance bottlenecks, 30	a care rable according to a specific filed it to fil	gringine potential potentiel (CK full CUOIIs
Ready				

5. Click the **OK** button to save the Modules filter settings.

 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Time-based Profiling (TBP) > Analyzing the TBP profile data > Call Graph view

Call Graph view

•

To navigate to Call Graph view, do any of the following:

 In CodeXL Explorer pane, double-click on Call Graph node in the profile session tree entry created for this profile run.

The **Call Graph** view is displayed with Callstack samples collected for the profiled application.

In **Overview** page, right-click on the **5 Hottest Functions** table entry, and select **Display in Call Graph View**.

The **Functions** view is displayed with Callstack samples collected for the profiled application.

In Functions view, select a function from the Functions table, and select Display in Call Graph View

This displays the Call Graph constructed from the callstack samples collected from the profiled application.

2. Click on the **Display: All Modules** link to enable **Display system modules**.

This displays the callstack samples collected from the system modules.

- 3. Click the **OK** button to save the Display filter settings.
- 4. Click on the **Process** combo box to select the process for which you want to view the Call Graph.

AMDTClassicMatMul - CodeXL Profile Mode (Cl File Edit View Debug Profile Frame Analy Vor Image: Solution of the state of the s	PU: Time-based Sampling) sis <u>A</u> nalyze <u>T</u> ools <u>W</u> indow <u>H</u> elp					-	
CodeXL Explorer 🗗 🗶	d Apr-07-2016_15-53-25 (CPU	I: Time-based Sampling)	×				
	🕘 Profile Overview 📕 Call Graph 🛛						
 AMDTClassicMatMul Profile Mode (CPU: O CPU: Time-based Sampling Apr-07-2016_15-50-43 	Functions	Display: <u>All Modules</u>		Process: AMDTC	assicMatMul.exe	(9964) 🔻 Monitored event:	٧
 Apr-07-2016_15-50-56 Apr-07-2016_15-53-25 Overview 	Function (47 functions) _scrt_common_main_seh main classic_multiply_matrices(void)	Self Samples	Deep Samples 512 508 502	% of Deep Samples 96.42% 95.67% 94.54%	No. of Paths 7 3 2	exe_common.inl(210) amdtclassicmatmul.cpp(160)	Modul / AMDT(AMDT(
Modules Call Graph Functions	initialize_matrices(void) rand qsort_s RtlSetCurrentTransaction	6 2 5	22 15 14 6	4.14% 2.82% 2.64% 1.13%	10 8 7 5	amdtclassicmatmul.cpp(90) amdtclassicmatmul.cpp(63)	AMDTC AMDTC ucrtbas ucrtbas ntdll.dl
	BaseThreadInitThunk LdrSetAopCompatDIIRedirectionCall		3 trices(void)	1.13% 0.56%	3	-	kernel3 ntdll.dl ` >
	Parents Samples main 502	% of samples Modul 100.00% AMDT	e ClassicMatM	Self + children [self] <	Sam 502	ples % of samples Modu 100.00% AMDT	le ClassicM
Properties 🗗 🗙	Dully.						
CPU: Profile Session	Paths	120 10 11 802	20	Show Call Graph sel			10
Profile Type Time-based Sampling Session Name Apr-07-2016_15-53-25 Executable Path: C: \ProgramData\AMD\CodeXL [Examples\ClassidMatMul/Release Apr-07-2016_15-53-25	Function _scrt_common_main_seh main classic_multiply_matrices(vo RtlSetCurrentTransaction		Downstream sa 500 500 2	mples % of Downst 99.60% 99.60% 99.60% 0.40%	ream samples	Module AMDTClassicMatMul.exe AMDTClassicMatMul.exe AMDTClassicMatMul.exe ntdll.dll	,
Working Directory: C:\ProgramData\AMD\CodeXL \Examples\ClassidMatMul\Release Profile Scope: Single Application Profile Start Time Apr-07-2016_15-53-25	RtlSetCurrentTransaction Drill down through call paths to find po	tential bottleneck functic	2 ns downstream fr	0.40%		ntdll.dll	





Help > Tutorials > CPU Profiler Tutorial > Analysis with Time-based Profiling (TBP) > Analyzing the TBP profile data > Source Code view

Source Code view

- To navigate to Call Graph view, do any of the following:

 In the **Overview** page, right-click on the **5 Hottest Functions** table entry, and select **Open Source Code**.
 This displays the **Source Code** view for the selected function.
 - In the **Functions** view, select a function from the **Functions** table, right-click and select **Open Source Code** from the context menu.
 - This displays the **Source Code** view for the selected function.
 - In **Call Graph** view, select a function from the **Functions** table, right-click and select **Open Source Code** from the context menu.
 - This displays the **Source Code** view for the selected function.
- 2. To view the sources for the some other function, click on the **Function** combo-box, and select the function for which you want to view the source code.

AMDTClassicMatMul - CodeXL Profile Mode (Cl Eile Edit View Debug Profile Frame Analy					- 0	Х	
🏷 🔿 🕶 🔟 🖉 🕨 Þ.Local 💌 📗							
CodeXL Explorer 🛛 🗗 🗙	Ŏ	Apr-07	-2016_15-53-25 (CPU: Time-based Sampling)				
🥥 🖾	Profile	Overview	🚊 Call Graph 🗵 🛛 🗄Release \AMDTClassicMatMul.exe - Source/Disassembly 🗵				
🗙 👍 AMDTClassicMatMul Profile Mode (CPU:	c:\programdata\amd\codex\\examples\classicmatmul\amdtclassicmatmul.cpp						
 <u>(PU: Time-based Sampling</u> <u>()</u> <u>()</u> Apr-07-2016_15-50-43) - 0x13c120e] : classic_multiply_matrices(void) ▼ Display: <u>All Data, All Modules</u>		Hotspot Indicator Tim	er 🔻	
> 👌 Apr-07-2016_15-50-56	Line	Address	Source Code Code Byte	s Hotspot Samples	% of Hotspot Samples	Tin ^	
✓ ₫ Apr-07-2016_15-53-25	89		void classic_multiply_matrices()				
0 Overview	> 90	0x13c10f0	{				
Modules	91		// Multiply the two matrices				
 Call Graph Functions 	> 92	0x13c10f8	for (int i = 0 ; i < ROWS ; i++)				
 Functions Gr Source Code 	93		{				
AMDTClassicMatMul.exe	> 94	0x13c1100	for (int j = 0 ; j < COLUMNS ; j++)				
I AMDTClassicMatMul.exe	95		{				
	> 96	0x13c1116	float sum = 0.0 ;	4	0.80%	4	
	97		for (int k = 0 ; k < COLUMNS ; k++)				
	98		{			1	
	> 99	0x13c1122	sum = sum + matrix_a[i][k] * matrix_b[k][j] ;	476	94.82%	476	
	100	0 10 11 10	}	22	R any	22	
		0x13c11d9	matrix_r[i][j] = sum ;	22	4.38%	22	
	102		}				
	102 / / 103 } > 104 0x13c120a }						
Properties 🗗 🗙	104	UXT3CT2Ud	1				
	105		void improved_multiply_matrices()				
CPU: Profile Session	107		{				
Profile Type Time-based Sampling	108		// Multiply the two matrices				
Session Name Apr-07-2016_15-53-25	109		for (int i = 0; i < ROWS; i++)				
Executable Path: C:\ProgramData\AMD\CodeXL	110						
Examples (ClassicMatMul (Release IAMDTClassicMatMul.exe	111		for (int k = 0; k < COLUMNS; k++)				
Working Directory: C: \ProgramData\AMD\CodeXL [Examples\ClassicMatMul\Release	112 <		ł			>	
Profile Scope: Single Application	Color	0 Select a function from the drop-down list to display its source/disassembly lines with sample counts					
Profile Start Time Apr-07-2016_15-53-25	U Sele						
V							
Ready							





Help > Tutorials > CPU Profiler Tutorial > Analysis with Event-based Profiling (EBP)

Analysis with Event-based Profiling (EBP)

Event-based profiling (EBP) uses the hardware performance event counters to measure the number of specific kinds of events that occur during execution. Examples of events include processor clock cycles, retired instructions, data cache accesses, and data cache misses. The specific events to be measured are determined by the profile configuration that is used to set up data collection. This section comprises the following modules.

Collecting EBP profile data

<u>Changing the EBP profile collection options</u> <u>Analyzing the EBP profile data</u>





Help > Tutorials > CPU Profiler Tutorial > Analysis with Event-based Profiling (EBP) > Collecting EBP profile data

Collecting EBP profile data

To collect the EBP data:

- 1. Create a new CodeXL Project
- 2. In the toolbar, select **CPU:** Assess **Performance** from the profile configuration list.

This is a predefined Event-based profile configuration provided by CodeXL.

Prof	lle	F <u>r</u> ame Analysis	<u>A</u> nalyze	Tools	Window	ŀ
	<u>S</u> ta	art Profiling (AMDT	ClassicMat	Mul.exe)) F5	
	Att	tach to Process				
~	Pro	ofile Mode - CPU: /	Ass <mark>ess P</mark> erf	ormance	1	
88	Pa	use Data Collection	n		F6	
~	СР	U: <u>A</u> ssess Perform	ance			
	СР	U: <u>C</u> ache Line Utili	ization			
	СР	U: Custom Profile				
	СР	U: Instruction-bas	ed Samplin	g		
	СР	U: In <u>v</u> estigate Brar	nching			
	СР	U: Investigate Data	Access			
	СР	U: Investigate Inst	uction Ac	cess		
	СР	U: Investigate <u>L</u> 2 C	ache Acce	ss		
	СР	U: Time- <u>b</u> ased Sar	mpling			
	GP	U: Performance Co	ounters			
	Ap	plication Timeline	Trac <u>e</u>			
	Poy	<u>w</u> er Profiling				
	S <u>e</u> l	ect Power Profiling	g Counters			
	Pro	ofile Settings				

3. Set the profile session options as explained in <u>Set the</u> <u>CPU Profile Session settings</u>.

Click the Start ► icon in the toolbar, or select Profile > Start Profiling, as shown below.

<u>P</u> rof	file	F <u>r</u> ame Analysis	<u>A</u> nalyze	<u>T</u> ools	<u>W</u> indow		
	<u>S</u> ta	art Profiling (AMD	TCIassic Mat	tMul.exe)	F5		
	Att	tach to Process					
~	Pro	ofile Mode - CPU:	Assess Perf	ormance			
88	Pa	F6					
~	СР	U: <u>A</u> ssess Perform	ance				
	CP	U: <u>C</u> ache Line Util	ization				
	CPU: Custom Profile						
	СР	U: Instruction-bas	ed Samplin	g			
	CP	U: In <u>v</u> estigate Bra	n <mark>ching</mark>				
	CPU: Investigate Data Access						
	CPU: Investigate Instruction Access						
	CPU: Investigate <u>L</u> 2 Cache Access						
	CP	U: Time- <u>b</u> ased Sa	mpling				
	GP	U: Performa <u>n</u> ce C	ounters				
	Ap	plication Timeline	Trac <u>e</u>				
	Poy	<u>w</u> er Profiling					
	Sel	lect Power Profilin	g Counters				
	P <u>r</u> c	ofile Settings					

CodeXL begins data collection and launches the specified application program. The start icon has been clicked and is disabled while profiling is in progress. The console window is open, and the launched application is running in the console window. The status bar displays the profile duration while profiling is running; this is a percentage of completion of data translation during data translation phase.



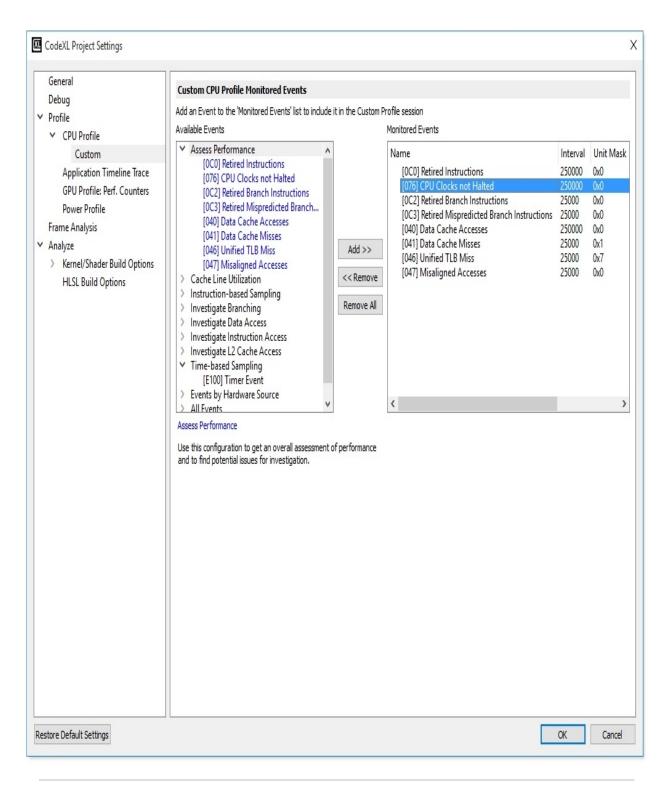


Help > Tutorials > CPU Profiler Tutorial > Analysis with Event-based Profiling (EBP) > Changing the EBP profile collection options

Changing the EBP profile collection options

CodeXL lets you modify the default profile collection settings. To do this:

- 1. Set the profile session options as explained in <u>Set the</u> <u>CPU Profile Session settings</u>.
- 2. By default, the sampling interval for TBP is 1 millisecond. To modify the sampling interval, click **Profile > Profile Settings** and click the **CPU Profile > Custom** node in the left pane tree in the **Edit CodeXL Project Settings** dialog box. The **Custom CPU Profile Monitored Events** page will be displayed.
- 3. Click the **Interval** field and set the required sampling interval.
- 4. Click the **OK** button to save the modified profile settings.







Help > Tutorials > CPU Profiler Tutorial > Analysis with Event-based Profiling (EBP) > Analyzing the EBP profile data

Analyzing the EBP profile data

This page explains how to analyze and navigate to the various profile views provided by the CodeXL CPU Profiler. Its modules are:

Overview page Modules view Functions view Call Graph view Source Code view





Help > Tutorials > CPU Profiler Tutorial > Analysis with Event-based Profiling (EBP) > Analyzing the EBP profile data > Overview page

Overview page

When data collection is complete, CodeXL processes the data and displays the results.

Initially, the **Profile Overview** is displayed with the following profile data:

- 5 Hottest Functions table
- 5 Hottest Modules table
- **Execution** section
- **Profile Details** section

🔟 AMDTClassicMatMul - CodeXL Profile Mode (CP						- 0	Х
File Edit View Debug Profile Frame Analys	s <u>A</u> nalyze <u>I</u> ools <u>W</u> indow <u>H</u> i	elp					
CodeXL Explorer	d Apr-07-2016_15-53-25 (CPU: Time	-based Sampling) 🔝 🔺 Ap	r-07-2016_16-05	i-10 (CPU: Assess Performance)	×		
	Profile Overview						
 AMDTClassicMatMul Profile Mode (CPU: O CPU: Time-based Sampling 	5 Hottest Functions		Display	y: <u>All Modules</u> Hotspot India	The second se	a Cache Accesses a Cache Accesses	•
 <i>o</i> Apr-07-2016_15-50-43 <i>o</i> Apr-07-2016_15-50-56 <i>o</i> Apr-07-2016_15-53-25 <i>o</i> Overview <i>o</i> Overview <i>o</i> Council Apr-07-2016_15-53-25 	Function classic_multiply_matrices(void) KeSynchronizeExecution initialize_matrices(void) goort_s	Samples % of Hotspot Sam) 3,061 84.16% 103 2.83% 77 2.12% 36 0.99%	AMDTCIa ntoskrnl. AMDTCIa ucrtbase.	ssicMatMul.exe dll	Misa CPU Reti Reti Dat	aligned Accesses J Clocks not Halted ired Instructions ired Branch Instructions ired Mispredicted Branch Instructions a Cache Misses fied TLB Miss	;
Functions Get Source Code	5 Hottest Processes	V		Hottest Modules	V		
AMDTClassicMatMul.exe Al CPU: Assess Performance Apr-07-2016_16-05-10 Overview Modules Call Graph	A STATE AND A STAT	amples % of Hotspot Sample 477 95.60% 50 4.40%		Module AMDTClassicMatMul.exe totoskrnl.exe ucrtbase.dll		% of Hotspot Samples 86.28% 5.94% 2.47%	>
I Functions I Froperties I X	Execution Target Path: C:\ProgramData\AMD\/ Working Directory: C:\ProgramData Profile Session Directory: C:\User: Command Line Arguments:	a \AMD \CodeXL \Examples \Classic	/atMul\Release		r-07-2016 <u>.</u>	_16-05-10	0
CPU: Profile Session	Environment Variables:						
Profile Type Assess Performance	CPU Affinity: 0xff	Profile Sco	pe: Single Applica	ation			
Session Name Apr-07-2016_16-05-10 Executable Path: C:\ProgramData\AMD\CodeXL \Examples\ClassicMatMul\Release \AMDTClassicMatMul.exe	Call Stack Sampling: True Profile Details	Call Stack 1	Information: So	ope - User mode, Depth - 32, Fi	requency -	per 1 samples, FPO - Off	
Working Directory: C:\ProgramData\AMD\CodeXL	Profile Session Type: Assess Perfor	rmance Monitored	Events				v
\Examples\ClassidMatMul\Release Profile Scope: Single Application Profile Start Time Apr-07-2016_16-05-10	() The 5 Hottest Functions usually i	indicate the most significant perfo	rmance bottlened	cks of your application. Select a	different H	lotspot Indicator to display hottest fi	uncti
Ready							

From the overview page, you can naviagte to various pages to analyze the profile data.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Event-based Profiling (EBP) > Analyzing the EBP profile data > Modules view

Modules view

 To navigate to the Modules view, do one of the following.
 In CodeXL Explorer pane, double-click on the Modules node in the profile session tree entry created for this profile run.

The **Modules** view is displayed.

In Overview page, right-click on the 5 Hottest Modules table entry, and select Display in Modules View.

The **Modules** view is displayed.

2. Click on the **Display: All Data, All Modules** link to set the Display Settings.

The **Display Settings** dialog box is displayed to let you set the various display settings.

CodeXL Explorer	₽×	d Apr-07-2016_15-53-25 (CPU	: Time-based Sampling) (Apr-07-2016_16-05-10	(CPU: Ass	sess Pe	erformance)	×			
(X	Profile Overview Mo	dules 🗵								
 AMDTClassicMatMul Profile M CPU: Time-based Sampli 		Processes		🗷 Display Settings	Х	<	🔁 Displa	y: <u>All Data, All Modu</u>	l <u>les</u> Display	by: Processe	es 🔻
> 👌 Apr-07-2016_15-50-4		Process PIE	DC accesses Mis	Columns		anch	Ret misp	branch DC miss	es TLB miss		
> d Apr-07-2016_15-50-5		AMDTClassicMatMul.exe 752	3,477 1	All Data	•		25	8,240			
 	5	conhost.exe 669	2 160 7	DC accesses			21	36	2		
Modules				Misalign access							
E Call Graph				CPU docks							
§ Functions				Ret inst							
🗸 📴 Source Code				Ret branch							
G AMDTClassic		Modules filtered by selected	processes	Ret misp branch							
 CPU: Assess Performance 		Module	Symbols DC acce	DC misses		st R	et branch	Ret misp branch	DC misses	TLB miss	^
Y Apr-07-2016_16-05-	-10	AMDTClassicMatMul.exe					866	13	8,206	least state	
Ø Overview Ø Modules		📲 ntoskrnl.exe	Not Loaded 216	TLB miss			06	13	43		
E Call Graph		📓 ucrtbase.dll	Not Loaded 90	General		3	15				
Functions		KernelBase.dll	Not Loaded 49	Display system modules			95	1			
a data		ntdll.dll 🕺 win32kfull.sys	Not Loaded 46 Not Loaded 24	Show percentages			56 20	2	2		1
		ntdll.dll	Not Loaded 24 Not Loaded 20	CPU Cores			32	4	3	1	
		win32kbase.sys	Not Loaded 14	All	^	5		2	7		
roperties	₽×	EX64.SYS	Not Loaded 8	Core 0 Core 1		5	В	4			
PU: Profile Session	٨	MTFS.sys	Not Loaded 6	Core 2 Core 3		1		1	2		
rofile Type Assess Performance		FLTMGR.SYS	Not Loaded 5	Core 4 Core 5	v	1					
ession Name Apr-07-2016_16-0		Cl.all	Not Loaded 4 Not Loaded 3			2			1		
xecutable Path: C:\ProgramData\AMD\		sysfer.dll	Not Loaded 2	Display data per:		2					
Examples Classic MatMul Release	ouc.	at msvcrt.dll	Not Loaded 2	O Core		6					
MDTClassidMatMul.exe		💒 atikmdag.sys	Not Loaded 2			1		1			V
/orking Directory: C:\ProgramData\AMI Examples\ClassicMatMul\Release	DICodeXL										
rofile Scope: Single Application				OK Ca	ancel						_
rofile Start Time Apr-07-2016_16-0.	5-10	Modules with a high sample (count usually indicate per	formance bottlenecks. Sort the tab	le accordir	ng to a	specific me	tric to highlight pote	ntial bottlene	ck modules	

3. Click **OK** button to save the Display filter settings.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Event-based Profiling (EBP) > Analyzing the EBP profile data > Functions view

Functions view

•

 To navigate to Functions view, do one of the following:

 In CodeXL Explorer pane, double-click on Functions node in the profile session tree entry created for this profile run.

The **Functions** view is displayed with function entries from all the profiled processes and the load modules.

In the **Overview** page, right-click on the **5 Hottest Modules** table entry, and select **Display in Functions View**.

The **Functions** view is displayed with function entries from the selected load module.

In the **Overview** page, right-click on the **5 Hottest Functions** table entry, and select **Display in Functions View**.

The **Functions** view is displayed with function entries from all the profiled processes and the load modules.

In the **Modules** view, select a Process from the **Processes** table, and select **Display in Functions View**. This displays the functions from the selected process and all its load modules.

In the **Modules** view, select a Module from the **Modules filtered by selected processes** table, and select **Display in Functions View**.

This displays the functions from the selected load module.

2. Click on the **Display: All Data, All Modules** link to set the display settings.

The **Display Settings** dialog box is displayed to let you set

the various display settings.

- 3. Click **OK** button to save the Display filter settings.
- 4. Click on the **modules shown, modules hidden** link to set the module filter settings to be used in the Functions view.

The **CodeXL Modules Filter** dialog box is displayed to let you set the module filter settings.

AMDTClassicMatMul - CodeXL Profile Mode					-	
ile Edit View Debug Profile FrameAn ा ● Local ▼ 11 ■	aiysis <u>A</u> naiyze <u>T</u> ools <u>W</u> indow <u>H</u> eip					
odeXL Explorer 🗗	K 👌 Apr-07-2016_15-53-25 (CPU: Time-based Sampling) 🛛	Apr-07-2016_16-05-10 (CPU: A	ssess Performance) 🗵			
	🖉 Profile Overview 🖉 Modules 🛛 🚺 Functions 🕻	3				
 MDTClassicMatMul Profile Mode (CPU: . OU: Time-based Sampling 	" Functions Displa	ay: <u>All Data, All Modules</u> <u>46 modu</u>	es shown, 0 modules hidden Proce	All Process	es	
> 👌 Apr-07-2016_15-50-43	Function	Module	DC accesses Misalign access	CPU clocks	Ret inst	Ret bran
> 👌 Apr-07-2016_15-50-56	classic_multiply_matrices(void)	AMDTClassicMatMul.exe	3,061	4,352	5,212	1,581
 	KeSynchronizeExecution	ntoskrnl.exe	103	229	76	197
Overview	initialize_matrices(void)	AMDTClassicMatMul.exe	77	45	127	285
🗌 Modules	a qsort_s	ucrtbase.dll	36	43	62	127
🖺 Call Graph	FIsGetValue	KernelBase.dll	34	30	77	161
Functions	RtlRestoreLastWin32Error	ntdll.dll	31	15	46	115
Y 🚱 Source Code	and rand	ucrtbase.dll	31	14	39	48
G AMDTClassicMatMul.exe	initialize_narrow_environment	ucrtbase.dll	17	18	52	121
✓ ▲ CPU: Assess Performance	GetLastError	KernelBase.dll	15	8	9	34
✓ ▲ Apr-07-2016_16-05-10	PfFileInfoNotify	ntoskrnl.exe	15	62	43	83
Overview	KeSetCoalescableTimer	ntoskrnl.exe	10		9	5
Modules	Figur EngUnmapFontFileFD	win32kfull.sys	8	7	12	24
Call Graph	RtllmageDirectoryEntryToData	ntdll.dll	7	9	8	24
III) management	KelnitializeEvent	ntoskrnl.exe	7	21	17	74
S Functions Call Graph	ExReleaseResourceLite	ntoskrnl.exe	7	17	17	39
	wcsstr	ucrtbase.dll	6		7	7
	RtlLookupFunctionEntry	ntdll.dll	6	18	23	71
	ExReleasePushLockExclusiveEx	ntoskrnl.exe	5		9	12
roperties 🗗		ntoskrnl.exe	5		2	9
	setjmpex	ntoskrnl.exe	5		6	14
PU: Profile Session	4 RtlActivateActivationContextUnsafeFast	ntdll.dll	4		10	19
rofile Type Assess Performance	RtlAcquireSRWLockShared	ntdll.dll	3	7	2	7
ession Name Apr-07-2016_16-05-10	MmUnlockPages	ntoskrnl.exe	3	3	1	1
ecutable Path: C:\ProgramData\AMD\CodeXL		ntoskrnl.exe	3	6	20	49
xamples \ClassicMatMul \Release	RtIAllocateHeap	ntdll.dll	2		3	3
MDTClassicMatMul.exe	FirstHookFunc	sysfer.dll	2		3	2
orking Directory: C:\ProgramData\AMD\CodeXL xamples\ClassicMatMul\Release	<					>
ofile Scope: Single Application		and a state of the set				C
ofile Start Time Apr-07-2016 16-05-10	Functions with a high sample count usually indicate performance	rmance bottlenecks. Sort the table ac	cording to a specific metric to highli	gnt potential bi	otteneck	TUNCTIONS

5. Click **OK** button to save the Modules filter settings.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Event-based Profiling (EBP) > Analyzing the EBP profile data > Call Graph view

Call Graph view

To navigate to Call Graph view, do one of the following:

 In the CodeXL Explorer pane, double-click on Call
 Graph node in the profile session tree entry created for this profile run.

The **Call Graph** view is displayed with Callstack samples collected for the profiled application.

- In the Overview page, right-click on the 5 Hottest
 Functions table entry, and select Display in Call
 Graph View. The Functions view is displayed with
 Callstack samples collected for the profiled application.
- In Functions view, select a function from the Functions table, and select Display in Call Graph View

This displays the Call Graph constructed from the callstack samples collected from the profiled application.

2. Click on the **Display: All Modules** link to enable **Display system modules**.

This displays the callstack samples collected from the system modules.

- 3. Click the **OK** button to save the Display filter settings.
- 4. Click on the **Process** combo box to select the process for which you want to view the Call Graph.

AMDTClassicMatMul - CodeXL Profile Mo	ode (CP	U: Assess Performance)								-		Х
<u>File E</u> dit <u>V</u> iew <u>D</u> ebug <u>P</u> rofile F <u>r</u> ame	Analys	is <u>A</u> nalyze <u>T</u> ools <u>W</u> indov	v <u>H</u> elp									
🖏 👸 🕶 🔟 🖉 🕨 Local 🔻 📗 🔳												
CodeXL Explorer	đ×	d Apr-07-2016_15-53-25 (CPU	I: Time-bas	ed Sampling) 🗵 🔺	Apr-07-2016_16-	05-10 (CPU: Assess Peri	formance)	X				
	XL		dules 🗵	§ Functions	E Call Graph	3						
🗸 🏟 AMDTClassicMatMul Profile Mode (CP												
✓ ♂ CPU: Time-based Sampling		Functions	[Display: <u>All Modules</u>		Process: AMDTClassic	latMul.ex	e(752) 🔻	Monitored event:	DC acces	ses	•
> 👌 Apr-07-2016_15-50-43		E .: (757 .:)		C.16.0	D C L	W (D)		D.(1 C	F 1	DC acces Misalign a	minute statements and	
> 👌 Apr-07-2016_15-50-56		Function (76 functions)		Self Samples			-	Paths So	burce File	CPU dod		
✓ ₫ Apr-07-2016_15-53-25		RtlSetCurrentTransaction BaseThreadInitThunk			3315 3315	99.61%	17 17			Ret inst Ret bran	ch	
Overview		_scrt_common_main_seh			3310	99.61% 99.46%	15	ev	common.inl(210	Ret misp		
Modules		main			3061	99.40% 91.98%	1	an	ndtclassicmatmul	TLB miss	5	
Call Graph Sunctions		classic_multiply_matrices	(void)	3061	3061	91.98%	1		ndtclassicmatmul.		AMD	Γ
Functions Geo Source Code		initialize_matrices(void)	(roid)	77	245	7.36%	11		ndtclassicmatmul.		AMD	
G AMDTClassicMatMul.exe		rand		31	153	4.60%	9			111.7	ucrtba	
 A CPU: Assess Performance 	•	qsort_s		36	78	2.34%	3				ucrtba	a
✓ ▲ Apr-07-2016_16-05-10		FlsGetValue		34	34	1.02%	2				Kerne	50
0 Overview		<									>	
Modules			<i>(</i>									
📕 Call Graph		Immediate Parents and Children	V					V				_
👔 Functions		Parents	Samples	% of samples Modu		Self + children		Samples	% of samples	Modul	le	^
		RtlSetCurrentTransaction	3315	100.00% ntdll.d	I	BaseThreadInitTl		3315	100.00%	kernel3		
						RtlSetCurrentTra	nsaction	3315	100.00%	ntdll.d	1	v
						<					>	
Properties	₽ X											
CPU: Profile Session	۸	Paths				Show Call Graph sel	ection pa	th				
Profile Type Assess Performance		Function		Self samples	Downstream sa	mples % of Downst	ream san	nples M	lodule			
Session Name Apr-07-2016_16-05-10		> RtlSetCurrentTransaction			3315	100.00%		nt	tdll.dll			
Executable Path: C:\ProgramData\AMD\CodeXL \Examples\ClassidMatMul\Release \AMDTClassidMatMul.exe												
Working Directory: C: \ProgramData\AMD\CodeXL \Examples\ClassicMatMul\Release												
Profile Scope: Single Application		() Drill down through call pat	hs to find p	otential bottleneck function	ons downstream fr	om your top level code						
Profile Start Time Apr-07-2016_16-05-10	v	~										
Ready	Ť											





Help > Tutorials > CPU Profiler Tutorial > Analysis with Event-based Profiling (EBP) > Analyzing the EBP profile data > Source Code view

Source Code view

- To navigate to Call Graph view, do one of the following:

 In the **Overview** page, right-click on the **5 Hottest Functions** table entry, and select **Open Source Code**.
 This displays the **Source Code** view for the selected function.
 - In the **Functions** view, select a function from the **Functions** table, right-click and select **Open Source Code** from the context menu.

This displays the **Source Code** view for the selected function.

In the **Call Graph** view, select a function from the **Functions** table, right-click and select **Open Source Code** from the context menu.

This displays the **Source Code** view for the selected function.

2. To view the sources for the some other function, click on the **Function** combo-box, and select the function for which you want to view the source code.

File Edit View Debu	g Profile Analyze Too	ls Windo	v Help						
NO-0 > 1		5	000	000					
CodeXL Explorer	₽ ×	A	S	ep-02-2014_04-30-43 (CPU: Assess Perfi	ormance)	X			
(0 Profile	Overview	C:\CodeXL\classic.exe - Source/Dis	assembly 🗵				
4 🏚 classic Profile Moo		c:\source\	samples\class	ic\classic.cpp					
▲ ♂ CPU: Time-base ▷ ♂ Sep-02-2014		Function	[0x401000	- 0x401060] : initialize_matrices(void)	Display: All Data, All N	1odules	Hotspot II	ndicator Data cache acce	isses 🔻
▷ 👌 Sep-02-2014	_04-26-20		[0x401000	- 0x401060] : initialize matrices(void)					
 A CPU: Assess Per A Sep-02-201 		Line		- 0x401195] : multiply_matrices(void) - 0x4012a9] : rand		Code Bytes	Hotspot Samples	% of Hotspot Samples	DC accesses DC mis *
Sep-02-20 Overvie	-	35	and the second se	- 0x401a11] : _getptd					
🔞 Module	S		0x401000	{					
🖺 Call Gra		37		// Define initial contents of the mat	trices				
👔 Function 🔺 🕞 Source (▷ 38	0x401003	for (int i = 0 ; i < ROWS ; i++) {					
Ge class		▷ 39	0x401010	for (int j = 0 ; j < COLUMNS ; j++			2		
> A Sep-02-2014	_07-01-45	▷ 40	0x401015	matrix_a[i][j] = (float) rand() /			27	46.55%	27
A Sep-03-2014			0x40102e	matrix_b[i][j] = (float) rand() /	RAND_MAX;		8	13.79%	8
A Sep-03-2014	00-51-43		0x40103b	matrix_r[i][j] = 0.0 ;			23	39.66%	23
		43		}					
		44		}					
			0x40105e	}					
Properties	5 X	46		······································					
CPU Profile Session		47		void print_result() ,					L
Sep-02-2014_04-30-43CF	U: Assess Performance	48		(// D					
Executable Path: C:\CodeXL\c	lassic.exe	49		// Print the result matrix					
Arguments:		50		for (int i = 0 ; i < ROWS ; i++) { for (int j = 0 ; j < COLUMNS ; j++	M				
Working Directory: C:\CodeXL		51							
Environment Variables:		52		fprintf(result_file, "%6.4f ", mat	ux_r(i)()));				
CPU Details: Family 0x10, Mod	el 10, 6 core(s)	55		; fprintf(result_file, "\n");					
Profile Scope: Single Applicatio		55		iprinu (result_nie, \n');					
Profile Duration: 10 seconds		55		1					
CPU Affinity: 0x3f		57		1					
Call Stack Information: Scope Frequency - per 1 samples, FF		58		void multiply_matrices()					
requercy sper i samples, Pr	0-01	•		"					Þ
		🚺 Sele	ect a function	from the drop-down list to display its sou	rce/disassembly lines with	sample counts			





Help > Tutorials > CPU Profiler Tutorial > Analysis with Instruction Based Sampling (IBS)

Analysis with Instruction Based Sampling (IBS)

Instruction Based Sampling (IBS) uses the hardware feature called IBS, to collect the IBS Fetch and IBS Op data. Collecting IBS profile data Changing the IBS profile collection options Analyzing the IBS profile data

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Instruction Based Sampling (IBS) > Collecting IBS profile data

Collecting IBS profile data

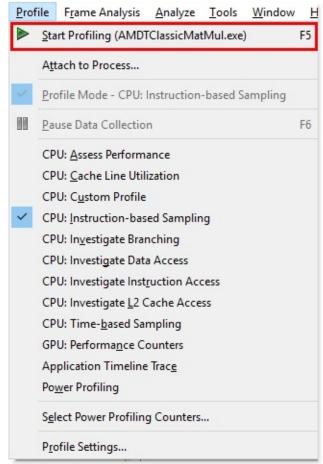
To collect the IBS data:

- 1. Create a new CodeXL Project.
- 2. In the toolbar, select **CPU: Instruction-based Sampling** from the profile configuration list, as shown in the following screenshot.

This is a predefined IBS profile configuration provided by CodeXL.

Prot	file	Frame Analysis	<u>A</u> nalyze	Tools	Window	H
	<u>S</u> ta	rt Profiling (AMDT	ClassicMat	tMul.exe)	F5
	A <u>t</u> t	ach to Process				
\checkmark	Pro	ofile Mode - CPU: I	nstruction	-based S	ampling	
88	Pa	use Data Collection	n			F6
	СР	U: <u>A</u> ssess Performa	ance			
	CP	U: <u>C</u> ache Line Utili	zation			
	СР	U: C <u>u</u> stom Profile				
~	СР	U: Instruction-bas	ed Samplir	ng		
	CP	U: In <u>v</u> estigate Brar	nching			
	CP	U: Investigate Data	Access			
	CP	U: Investigate Inst	uction Ac	cess		
	CP	U: Investigate <u>L</u> 2 C	ache Acce	SS		
	CP	U: Time- <u>b</u> ased Sar	mpling			
	GP	U: Performa <u>n</u> ce Co	ounters			
	Ap	plication Timeline	Trac <u>e</u>			
	Poy	<u>w</u> er Profiling				
	Sel	ect Power Profiling	g Counters			
	Pro	ofile Settings				

- 3. Set the profile session options as explained in <u>Set the</u> <u>CPU Profile Session settings</u>.
- Click the Start ▶ icon in the toolbar, or select Profile > Start Profiling, as shown below.



CodeXL begins data collection and launches the specified application program. The start icon is disabled until profiling is in progress. The console window is open, and the launched application is running in the console window. The status bar displays the profile duration while profiling is running; this is the percentage completion of data translation during data translation phase.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Instruction Based Sampling (IBS) > Changing the IBS profile collection options

Changing the IBS profile collection options

CodeXL lets you modify the default profile collection settings. To do this:

- 1. Set the profile session options as explained in <u>Set the</u> <u>CPU Profile Session Options</u>.
- 2. By default, the sampling interval for TBP is 1 millisecond. To modify the sampling interval, click **Profile > Profile Settings** and click on the **CPU Profile > Custom** node in the left pane tree. in the **Edit CodeXL Project Settings** dialog box. The **Custom CPU Profile Monitored Events** page will be displayed.
- 3. Click the **Interval** field and set the required sampling interval.
- 4. Click the **OK** button to save the modified profile settings.

CodeXL Project Settings							Х
General Debug ♥ Profile	Custom CPU Profile Monitored Events Add an Event to the 'Monitored Events' list to include Available Events	it in the Custom P	rofile session Monitored Events				
 CPU Profile Custom Application Timeline Trace GPU Profile: Perf. Counters Power Profile Frame Analysis Analyze Kernel/Shader Build Options HLSL Build Options 	Available Events Assess Performance Cache Line Utilization Instruction-based Sampling [F000] All IBS op samples Investigate Branching Investigate Data Access Investigate Instruction Access Investigate L2 Cache Access Time-based Sampling [E100] Timer Event Events by Hardware Source All Events	Add >> << Remove Remove All ut instruction TLB ructions.	Monitored Events Name [F100] All IBS op samples [F000] All IBS fetch samples [F000] All IBS fetch samples Verticity Verticity © Count ops dispatched O count dock cycles	Interval 250000 250000	Unit Mask Ox1	User Mode	Kerne
Restore Default Settings					OK	Ca	ancel





Help > Tutorials > CPU Profiler Tutorial > Analysis with Instruction Based Sampling (IBS) > Analyzing the IBS profile data

Analyzing the IBS profile data

This page explains how to analyze and navigate to the various profile views provided by the CodeXL CPU Profiler. It is comprised of the following modules.

Overview page Modules view Functions view Call Graph view Source Code view





Help > Tutorials > CPU Profiler Tutorial > Analysis with Instruction Based Sampling (IBS) > Analyzing the IBS profile data > Overview page

Overview page

When data collection is complete, CodeXL processes the data and displays the results.

Initially, the **Profile Overview** displays the following profile data.

- 5 Hottest Functions table
- 5 Hottest Modules table
- **Execution** section
- **Profile Details** section

AMDTClassicMatMul - CodeXL Profile Mode (C <u>File Edit View D</u> ebug <u>P</u> rofile F <u>r</u> ame Analy		łp				- 0	Х
🖏 讨 🕶 🔟 🕨 Local 🔻 📗 🔳		i.					
CodeXL Explorer & X	5 (CPU: Time-based Sampling) 🔟 🔺	Apr-07-2016_16-05-10 (CPU: Assess	Performance) 🗵	Apr-07-2016_1	6-14-19 (CF	PU: Instruction-based Sampling) 🛽	•
(1) (1)	Profile Overview						
✓ ▲ AMDTClassicMatMul Profile Mode (C ∧ ✓ ♂ CPU: Time-based Sampling ▲ ▲ 47 2006 15 50 40	5 Hottest Functions	Di	splay: <u>All Modules</u>	Hotspot Indicator	All IBS feto All IBS feto		•
 Apr-07-2016_15-50-43 Apr-07-2016_15-50-56 Apr-07-2016_15-53-25 Overview Modules Call Graph 	Function classic_multiply_matrices(void) classic_multiply_matrices(void) classic_multiply_matrices(void) classic_matrix classic_matrix classic_matrix classic_matrix classic_multiply_matrix	30 2.42% 15 1.21%	Module AMDTClassicMat ucrtbase.dll ntoskrnl.exe ucrtbase.dll ucrtbase.dll	/ul.exe	IBS fetch k IBS fetch a IBS fetch a IBS fetch a IBS ITLB hi IBS L1 ITLB IBS L1 L2 I IBS instruc	illed attempted soorpleted aborted it 3 misses (and L2 ITLB hits)	Ĵ
 Functions Gr Source Code 	5 Hottest Processes		5 Hottes	t Modules	103 1130 00		
AMDTClassicMatMul.exe ADTClassicMatMul.exe Apr-07-2016_16-05-10 Overview Modules Call Graph Functions Functions		mples % of Hotspot Samples 098 88.33% 5 11.67%	at ntos)TClassicMatMul.exe krnl.exe base.dll	952 84 68	% of Hotspot Samples 76.59% 6.76% 5.47% 2.09%	>
 Apr-07-2016_16-14-19 Overview 	Target Path: C:\ProgramData\AMD\C Working Directory: C:\ProgramData			Iul.exe			
Properties 🗗 🗙	Profile Session Directory: C:\Users			tMul_ProfilerOutput\Ap	or-07-2016_	_16-14-19	
CPU: Profile Session	Command Line Arguments:						
Profile Type Instruction-based Sampling	Environment Variables:						
Session Name Apr-07-2016_16-14-19	CPU Affinity: 0xff	Profile Scope:	Single Application				
Executable Path: C: \ProgramData\AMD\CodeXL \Examples\ClassicMatMul\Release \AMDTClassicMatMul.exe	Call Stack Sampling: True	Call Stack Inf	ormation: Scope -	User mode, Depth - 32	, Frequ <mark>enc</mark> y	y - per 1 samples, FPO - Off	
Working Directory: C:\ProgramData\AMD\CodeXL \Examples\ClassicMatMul\Release	Profile Details						v
Profile Scope: Single Application Profile Start Time Apr-07-2016_16-14-19	1 The 5 Hottest Functions usually in	ndicate the most significant performanc	e bottlenecks of you	ur application. Select a	different He	otspot Indicator to display hottest	functi
Ready							

From the overview page you can navigate to various pages to analyze the profile data.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Instruction Based Sampling (IBS) > Analyzing the IBS profile data > Modules view

Modules view

 To navigate to Modules view, do one of the following.
 In the CodeXL Explorer pane, double-click on the Modules node in the profile session tree entry created for this profile run.

The **Modules** view is displayed.

In the **Overview** page, right-click on the **5 Hottest Modules** table entry, and select **Display in Modules View**.

The **Modules** view is displayed.

2. Click on the **Display: All Data, All Modules** link to set the Display Settings.

The **Display Settings** dialog box is displayed to let you set the various display settings.

🔯 👸 🔹 💷 🙋 📄 Local 💌 🛽	₽×	5 (CPU: Time-based Sampling) 🗵	Apr-07-2016_16-	05-10 (CPU: Assess Performance) 🗵	A	.pr-07-2016_16-14	4-19 (CPU: Instruct	ion-based Sampling) 🗵
	X	Profile Overview Mo	dules 🗵					
 		Processes		🖾 Display Settings	×	Display: <u>/</u>	Il Data, All Modules	Display by: Processes
Overview		Process PID	IBS fetch IBS fetc	Columns	pm	np IBS fetch ab	ort IBS L1 ITLB	hit IBS ITLB L1M L2H
Modules		AMDTClassicMatMul.exe 429	6 1,098	All Data	•	3	1,097	
Call Graph Functions		conhost.exe 697	6 145	🗹 IBS fetch	٨	4	145	
 Functions Get Source Code 				🗹 IBS fetch killed	-			
G AMDTClassicM	atMul.exe			☑ IBS fetch attempt				
✓ ▲ CPU: Assess Performance	at manufactor	<		☑ IBS fetch comp				
× A Apr-07-2016_16-05-10				IBS fetch abort				
Overview		Modules filtered by selected	processes	IBS L 1 ITLB hit				
🔞 Modules		Module	Symbols IBS fe	IBS ITLB L 1M L 2H		IBS fetch comp	IBS fetch abort	IBS L1 ITLB hit IBS IT
🖺 Call Graph		AMDTClassicMatMul.exe	Loaded 952			951	1	952
Functions		ntoskrnl.exe	Not Loaded 84	IBS ITLB L 1M L 2M	*	83	1	932 84
CPU: Instruction-based S		ucrtbase.dll	Not Loaded 68	General		68		68
✓ ■ Apr-07-2016_16-14-1	19	atdl.dll	Not Loaded 26	Display system modules		26		26
 Overview Modules 		💒 win32kfull.sys	Not Loaded 22	Show percentages		21	1	22
Call Graph		ntdll.dll	Not Loaded 18	CPU Cores		18		18
Functions		win32kbase.sys £4 EX64.SYS	Not Loaded 17	All	A .	17		17
	•	💒 EX64.SYS 🙀 KernelBase.dll	Not Loaded 11 Not Loaded 9	Core 0 Core 1		10 9	1	11
operties	₽×	a hal.dll	Not Loaded 3	Core 2 Core 3		7		7
PU: Profile Session	٨	MTFS.sys	Not Loaded 4	Core 4 Core 5		4		4
ofile Type Instruction-based Sa	ampling	FLTMGR.SYS	Not Loaded 3		V	3		3
Apr-07-2016_16-14	-19	BHDrvx64.sys	Not Loaded 2	Display data per:		2		2
ecutable Path: C:\ProgramData\AMD\C xamples\ClassicMatMul\Release MDTClassicMatMul.exe	odeXL	<pre> # fileinfo.sys # gdi32.dll </pre>	Not Loaded 2 Not Loaded 2	Core		2 2		2 2
orking Directory: C:\ProgramData\AMD xamples\ClassicMatMul\Release	CodeXL			OK Cance	el			,
ofile Scope: Single Application								

3. Click the **OK** button to save the Display filter settings.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Instruction Based Sampling (IBS) > Analyzing the IBS profile data > Functions view

Functions view

•

 To navigate to Functions view, do one of the following.
 In CodeXL Explorer pane, double-click on **Functions** node in the profile session tree entry created for this profile run.

The **Functions** view is displayed with function entries from all the profiled processes and the load modules.

In the **Overview** page, right-click on the **5 Hottest Modules** table entry, and select **Display in Functions View**.

The **Functions** view is displayed with function entries from the selected load module.

In the **Overview** page, right-click on the **5 Hottest Functions** table entry, and select **Display in Functions View**.

The **Functions** view is displayed with function entries from all the profiled processes and the load modules.

In the **Modules** view, select a Process from the **Processes** table, and select **Display in Functions View**. This displays the functions from the selected process and all its load modules.

In the **Modules** view, select a Module from the **Modules filtered by selected processes** table, and select **Display in Functions View**.

This displays the functions from the selected load module.

2. Click on the **Display: All Data, All Modules** link to set the display settings.

The **Display Settings** dialog box is displayed to let you set the various display settings.

- 3. Click the **OK** button to save the Display filter settings.
- 4. Click on the **modules shown, modules hidden** link to set the module filter settings to be used in the Functions view.

The **CodeXL Modules Filter** dialog box is displayed to let you set the module filter settings.

deXL Explorer & X	5 (CPU: Time-based Sampling) 🗵 🛛 🛕	Apr-07-2016_16-05-10 (CPU: Assess Per	formance) 🗵 🔳 Apr-07-20	16_16-14-19 (CPU: Instruction-bas	ed Sampling) 🗵
	Profile Overview Modules) 🚯 Functions 🗵			
 À pr-07-2016_15-50-56 À pr-07-2016_15-53-25 Overview Modules Call Graph Functions Functions Fource Code AMDTClassicMatMul.exe CPU: Assess Performance Apr-07-2016_16-05-10 Overview Modules Call Graph Functions CPU: Instruction-based Sampling Functions Functions Functions Functions CPU: Instruction-based Sampling Functions Functions Functions Call Graph Functions Functions V CPU: Instruction-based Sampling Functions Functions Functions Functions Functions 	Function Function classic_multiply_matrices(void) qsort_s 4 PfFileInfoNotify initialize_narrow_environment a rand 4 RtILookupFunctionEntry 4 KeSynchronizeExecution R RtIRestoreLastWin32Error wcsstr 4 ExReleaseResourceLite FIsGetValue R RtIImageDirectoryEntryToData initialize_matrices(void) _report_gsfailure 4 EngUnmapFontFileFD 4 RtIActivateActivationContextUn: 4 HalRequestIpi 4 ExReleasePushLockExclusiveEx 4 RtIFreeHeap 4 GdiProcessCallout 4 HmgNextOwned 4 memcpy	Display: <u>All Data, All Modu</u> Modules CodeXL Modules Filter Modules of System-wide Profiling(2 Pro Modules AMDTClassicMatMul.exe MemelBase.dll ADDrxx64.sys KernelBase.dll MemerelBase.dll Mem	e IBS fetC	ers\ ers\ ers\	IBS fetch coi
ssion Name Apr-07-2016_16-14-19	⁶⁴ memcmp ⁶⁴ memset	ntoskrn		3	3
ecutable Path: C:\ProgramData\AMD\CodeXL camples\ClassicMatMul\Release MDTClassicMatMul.exe orking Directory: C:\ProgramData\AMD\CodeXL camples\ClassicMatMul\Release	*** memset GetLastError ** RtIAllocateHeap	ntoskrn KernelB ntdli.dli	ase.dll 2	3 2 2	3 2 2

5. Click the **OK** button to save the Modules filter settings.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Instruction Based Sampling (IBS) > Analyzing the IBS profile data > Call Graph view

Call Graph view

•

 To navigate to Call Graph view, do one of the following.
 In the CodeXL Explorer pane, double-click on the Call Graph node in the profile session tree entry created for this profile run.

The **Call Graph** view is displayed with Callstack samples collected for the profiled application.

In the **Overview** page, right-click on the **5 Hottest Functions** table entry, and select **Display in Call Graph View**.

The **Functions** view is displayed with Callstack samples collected for the profiled application.

In the **Functions** view, select a function from the **Functions** table, and select **Display in Call Graph View**

This displays the Call Graph constructed from the callstack samples collected from the profiled application.

2. Click on the **Display: All Modules** link to enable the **Display system modules**.

This displays the callstack samples collected from the system modules.

- 3. Click the **OK** button to save the Display filter settings.
- 4. Click on the **Process** combo box to select the process for which you want to view the Call Graph.

AMDTClassicMatMul - CodeXL Profile Mode File Edit View Debug Profile Frame An	(CPU: Instruction-based Sampling) alysis <u>A</u> nalyze <u>T</u> ools <u>W</u> indow <u>H</u> elp					-	
	🗙 5 (CPU: Time-based Sampling) 🗵 🛛 🔺 Apr	-07-2016_16-05-10 (CP	U: Assess Perform	ance) 🗵 🔳 Apr-	07-2016_16-14-19) (CPU: Instruction-based Samp	ling) 🗵 🛛
(a) 🔊	Profile Overview 🔞 Modules 🗵	🗴 Functions 🗵	📕 Call Graph	×			
 	Functions	Display: <u>All Modules</u>		Process: AMDTC	assicMatMul.exe(4		BS fetch 🔹 BS fetch
 Overview Modules Call Graph Functions Functions Source Code AMDTClassicMatMul.exe Apr-07-2016_16-05-10 Overview Modules Call Graph Functions CPU: Instruction-based Sampling CPU: Instruction-based Sampling Modules COURTING Overview Modules Call Graph Functions Call Graph 	Function (60 functions) RtlSetCurrentTransaction BaseThreadInitThunk _scrt_common_main_seh main classic_multiply_matrices(void) initialize_matrices(void) rand qsort_s RttRestoreLastWin32Error Immediate Parents and Children of Function Parents Samples main 947	947 32 10 18 11 : classic_multiply_m % of samples Mode		% of Deep Samples 99.14% 99.14% 98.86% 90.19% 90.19% 8.57% 5.33% 3.05% 1.05% 1.05% Self + children [self]	1 a 1 a 12 a 9 4 2	exe_common.inl(210) amdtclassicmatmul.cpp(160) amdtclassicmatmul.cpp(90) amdtclassicmatmul.cpp(63)	AMDTCI AMDTCI ucrtbase ucrtbase ntdll.dll
i Functions	× .			<			
•	A Paths			Show Call Graph s	election path		
Profile Type Instruction-based Sampling	Function	Self samples	Downstream s	amples % of Down	tream samples	Module	
Session Name Apr-07-2016_16-14-19	✓ RtlSetCurrentTransaction		947	100.00%		ntdll.dll	
xecutable Path: C:\ProgramData\AMD\CodeXL	✓ RtlSetCurrentTransaction		947	100.00%		ntdll.dll	
Examples \ClassicMatMul \Release AMDTClassicMatMul.exe	✓ BaseThreadInitThunk		947	100.00%		kernel32.dll	
	✓ _scrt_common_main_se	h	947	100.00%		AMDTClassicMatMul.exe	
Norking Directory: C:\ProgramData\AMD\CodeXL Examples\ClassicMatMul\Release	✓ main		947	100.00%		AMDTClassicMatMul.exe	
Yrofile Scope: Single Application Yrofile Start Time Apr-07-2016_16-14-19	Drill down through call paths to find p	otential bottleneck funct	ions downstream l	from your top level code			





Help > Tutorials > CPU Profiler Tutorial > Analysis with Instruction Based Sampling (IBS) > Analyzing the IBS profile data > Source Code view

Source Code view

- To navigate to Call Graph view, do one of the following.
 In the **Overview** page, right-click on the **5 Hottest Functions** table entry, and select **Open Source Code**.

 This displays the **Source Code** view for the selected function.
 - In the **Functions** view, select a function from the **Functions** table, right-click and select **Open Source Code** from the context menu.
 - This displays the $\ensuremath{\textbf{Source Code}}$ view for the selected function.
 - In the **Call Graph** view, select a function from the **Functions** table, right-click and select **Open Source Code** from the context menu.
 - This displays the **Source Code** view for the selected function.
- 2. To view the sources for the some other function, click on the **Function** combo-box, and select the function for which you want to view the source code.

AMDTClassicMatMul - CodeXL Profile Mode (C	- PU: Instruction-based Sampling)	- 0 X
<u>File Edit View Debug Profile Frame Analy</u>	rsis <u>A</u> nalyze <u>T</u> ools <u>W</u> indow <u>H</u> elp	
🏷 🔿 🕶 🙋 🕨 Local 🔻 📗 🔳		
CodeXL Explorer 🗗 🗙	5 (CPU: Time-based Sampling) 🔟 🔺 Apr-07-2016_16-05-10 (CPU: Assess Performance) 🔟 🗐 Apr-07-2016_16-14-19 (CPU: Instruction-base	d Sampling) 🔀 🔳
💭 🔍	🔘 Profile Overview 🛞 Modules 🗶 🏦 Functions 🗵 🖹 Call Graph 🗵 🕼Release\AMDTClassicMatMul.exe - Source/Disassem	
Overview		
Modules	c: programdata amd codexl examples dassicmatmul amdtdassicmatmul.cpp	
📕 Call Graph	Function: [0x1510f0 - 0x15120e] : dassic_multiply_matrices(void) Display: <u>All Data, All Module</u> <u>Hotspot Indicator</u> All IBS fetch samples	•
§ Functions	[0x151080 - 0x1510e5] : initialize_matrices(void) Line [0x1510f0 - 0x15120e] : dassic_multiply_matrices(void)	Code Bytes 🔺
🗸 🚱 Source Code	[0x1513e0 - 0x1516d6] : main	
G AMDTClassicMatMul.exe	89 [0x1518bf - 0x151a3b] : _scrt_common_main_seh > 90 0x1510f0 {	
✓ ▲ CPU: Assess Performance	91 // Multiply the two matrices	
 Apr-07-2016_16-05-10 	> 92 0x1510f8 for (int i = 0; i < ROWS; i++)	
Overview	93 {	
🛞 Modules	> 94 0x151100 for (int j = 0; j < COLUMNS; j++)	
🖺 Call Graph	95 {	
Functions	> 96 0x151116 float sum = 0.0;	
CPU: Instruction-based Sampling	97 for (int k = 0 ; k < COLUMNS ; k++)	
 Apr-07-2016_16-14-19 Overview 	98 {	
Modules	99 0x151122 sum = sum + matrix_a(i][k] * matrix_b[k](j];	
Call Graph	100 }	
Functions	> 101 0x1511d9 matrix_r[i][j] = sum;	
V G Source Code	102 }	
G AMDTClassicMatMul.exe	103 }	
	> 104 0x15120a }	
Properties 🗗 🗙	105 106 void improved_multiply_matrices()	
CPU: Profile Session	107 {	
Profile Type Instruction-based Sampling	107 I Multiply the two matrices	
Session Name Apr-07-2016_16-14-19	109 for (int i = 0; i < ROWS; i++)	
executable Path: C:\ProgramData\AMD\CodeXL	110 {	
Examples (ClassicMatMul \Release \AMDTClassicMatMul.exe	111 for (int k = 0; k < COLUMNS; k++)	
, Working Directory: C:\ProgramData\AMD\CodeXL Examples\ClassicMatMul\Release	<pre>112 {</pre>	>
Profile Scope: Single Application	Calast a function from the day late to deal with any cale of the second by the second second	
Profile Start Time Apr-07-2016_16-14-19	() Select a function from the drop-down list to display its source/disassembly lines with sample counts	
Ready		





Help > Tutorials > CPU Profiler Tutorial > Analysis with Cache Line Utilization (CLU)

Analysis with Cache Line Utilization (CLU)

Cache Line Utilization (CLU) uses the Load and Store Instruction-Based Sampling (IBS) records to provide a measure of how efficiently an application utilizes the L1 data cache.

<u>Collecting CLU profile data</u>

Changing the CLU profile collection options

Analyzing the CLU profile data

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Cache Line Utilization (CLU) > Collecting CLU profile data

Collecting CLU profile data

To collect the CLU data, perform the following steps:

- 1. First Create a new CodeXL Project
- 2. In the toolbar, select **CPU: Cache Line Utilization** from the profile configuration list, as shown in the following figure. This is a predefined Cache Line Utilization profile configuration provided by CodeXL.

Profile		Frame Analysis	Analyze	Tools	Window	Н	
	Start Profiling (AMDTClassicMatMul.exe) F5 Attach to Process Profile Mode - CPU: Cache Line Utilization						
~							
88	Pa	use Data Collection	F6				
	СР	U: Assess Perform	ance				
~	СР	U: Cache Line Utili	zation				
	CP	U: Custom Profile					
	СР	U: Instruction-bas	ed Samplin	g			
	CPU: Investigate Branching						
	CPU: Investigate Data Access						
	CPU: Investigate Instruction Access						
	CPU: Investigate L2 Cache Access						
	CPU: Time-based Sampling						
	GPU: Performance Counters						
	Ар	plication Timeline	Trace				
	Po	wer Profiling					
	Sel	lect Power Profiling	g Counters				
	Profile Settings						

- 3. Set the profile session options as explained in <u>Set the CPU</u> <u>Profile Session settings</u>
- 4. Click the Start > icon in the toolbar or select **Profile** >

Start Profiling as shown below.

Profile		Frame Analysis Analyze To	ools	Window H			
▶	Sta	rt Profiling (AMDTClassicMatMu	ul.exe)) F5			
	Att	tach to Process					
~	Profile Mode - CPU: Cache Line Utilization						
88	Pau	F6					
	СР	U: Assess Performance					
~	CP	U: Cache Line Utilization					
	CP	U: Custom Profile					
	СР	U: Instruction-based Sampling					
	СР	U: Investigate Branching					
	CPU: Investigate Data Access						
	CPU: Investigate Instruction Access						
	CPU: Investigate L2 Cache Access						
	CP	U: Time-based Sampling					
	GP	U: Performance Counters					
	Ар	plication Timeline Trace					
	Po	wer Profiling					
	Sel	lect Power Profiling Counters					
	Pro	ofile Settings					

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Cache Line Utilization (CLU) > Changing the CLU profile collection options

Changing the CLU profile collection options

CodeXL lets you modify the default profile collection settings. To do this:

- 1. Set the profile session options as explained in <u>Set the</u> <u>CPU Profile Session settings</u>.
- 2. By default, the sampling interval for TBP is 1 millisecond. To modify the sampling interval, click **Profile > Profile Settings** and click on the **CPU Profile > Custom** node in the left pane tree. in the **Edit CodeXL Project Settings** dialog box. The **Custom CPU Profile Monitored Events** page will be displayed.
- 3. Click the **Interval** field and set the required sampling interval.
- 4. Click the **OK** button to save the modified profile settings.

CodeXL Project Settings				Х
General Debug ✔ Profile	Custom CPU Profile Monitored Events Add an Event to the 'Monitored Events' list to include i	t in the Custom Profile session		
	Available Events Monitored Events			
 CPU Profile Custom Application Timeline Trace GPU Profile: Perf. Counters Power Profile Frame Analysis Analyze Kernel/Shader Build Options HLSL Build Options 	Available Events Available Events Cache Line Utilization [FF00] Cache Line Utilization Instruction-based Sampling [F00] All IBS op samples [F00] All IBS fetch samples Investigate Branching Investigate Data Access Investigate Instruction Access Investigate L2 Cache Access Investigate L2 Cache Access Time-based Sampling [E100] Timer Event Events by Hardware Source All Events [FF00] Cache Line Utilization Cache line utilization provides information about the to cache lines in L1 Data cache which helps to find out to data locality and data access pattern.	Add >> << Remove All Event Settings	Interval Unit Mask User Mode	×
Restore Default Settings			OK Car	icel

 $\ensuremath{\textcircled{\sc c}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Cache Line Utilization (CLU) > Analyzing the CLU profile data

Analyzing the CLU profile data

This page explains how to analyze and navigate to the various profile views provided by the CodeXL CPU Profiler.

Overview page Overview page Modules view Modules view Functions view Functions view Source Code view Source Code view





Help > Tutorials > CPU Profiler Tutorial > Analysis with Cache Line Utilization (CLU) > Analyzing the CLU profile data > Overview page

Overview page

When data collection is complete, CodeXL processes the raw profile data and displays the results.

Initially the **Profile Overview** is displayed. It displays the following profile data:

- 5 Hottest Functions table
- 5 Hottest Modules table
- 5 Hottest Processes table
- Execution section
- · Profile Details section

AMDTClassicMatMul - CodeXL Profile	Mode (CP	J: Cache Line Utilization) — 🗌 🔿	K					
<u>File Edit View D</u> ebug <u>Profile</u> F <u>r</u> an	me Analys	s <u>A</u> nalyze <u>I</u> ools <u>W</u> indow <u>H</u> elp						
🖏 👩 🕶 🔟 🖉 🕨 Local 💌 📗								
CodeXL Explorer	₽×	III Apr-07-2016_16-22-43 (CPU: Cache Line Utilization)						
	X	Profile Overview						
AMDTClassicMatMul Profile Mode A G CPU: Time-based Sampling A CPU: Assess Performance	CPU:	5 Hottest Functions Display: <u>All Modules</u> Hotspot Indicator Cache Line Utilization Percentage Cache Line Utilization Percentage						
 CPU: Instruction-based Sampling CPU: Cache Line Utilization CPU: Cache Line Utilization Apr-07-2016_16-22-43 Overview Modules 	I	Function Cache Line Utilization Percentage (Higher is better) Module Line Boundary Crossings Bytes/L1 Eviction ** KeLeaveCriticalRegion 3.12% ntoskrnl.exe Accesses/L1 Eviction ** EX64.SYS10xf80103aba192 1.56% EX64.SYS L1 Evictions ** RtlLookupFunctionEntry 1.56% ntdll.dll Bytes Accessed ** Win32FreeToPagedLookasideListImpl 1.56% win32kfull.sys	~					
Functions		5 Hottest Processes 5 Hottest Modules						
		Process PID Cache Line Utilization Percentage (Higher is conhost.exe 4412 8.67% AMDTClassicMatMul.exe 8824 6.29% 4412 8.67% Image: Conhost.exe 4412 8.67% 1411.dll 5.86% Image: Conhost.exe 4412 6.29% 1411.dll 5.73% Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe 1411.dll 5.73% Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe 4412 6.29% Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe 4412 6.29% Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe 4412 6.29% Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image: Conhost.exe Image:	~					
		Target Path: C: ProgramData (AMD (CodeXL Examples (ClassicMatMul/Release (AMDTClassicMatMul.exe						
		Working Directory: C: (ProgramData\AMD)CodeXL/Examples (ClassidMalMul/Release						
Properties	đ×	Profile Session Directory: C: {Users\salgrana \AppData\Roaming \AMD\CodeXL \AMDTClassidMatMul_ProfilerOutput\Apr-07-2016_16-22-43 Command Line Arguments:						
CPU: Profile Session	٨	Environment Variables:						
Profile Type Cache Line Utilization		CPU Affinity: 0xff Profile Scope: Single Application						
Session Name Apr-07-2016_16-22-43		Call Stack Sampling: False						
Executable Path: C:\ProgramData\AMD\CodeXL \Examples\ClassicMatMul\Release \AMDTClassicMatMul.exe		Profile Details						
Working Directory: C:\ProgramData\AMD\CodeXI \Examples\ClassicMatMul\Release		Profile Session Type: Cache Line Utilization Monitored Events	1					
Profile Scope: Single Application Profile Start Time Apr-07-2016_16-22-43	~	1 The 5 Hottest Functions indicate the functions having lowest utilization (read/write) of L1 data cache lines between evictions. Select a different Hotspot Indicator to d						
Ready								

From the overview page the user can navigate to various pages to analyze the profile data.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Cache Line Utilization (CLU) > Analyzing the CLU profile data > Modules view

Modules view

- 1. To navigate to Modules view, do any of the following:
 - In CodeXL Explorer pane, double click on Modules node in the profile session tree entry created for this profile run. The Modules view is displayed.
 - In Overview page, right click on 5 Hottest Modules table entry and select Display in Modules View. The Modules view is displayed.
- 2. Click on **Display: All Data, All Modules** link to set the Display Settings. The **Display Settings** dialog box is displayed to enable the users to set the various display settings.

🚫 🚫 🔻 🛄 🖳 🕨 Local 🔻 📗 🔳										
CodeXL Explorer & X	Apr-07-2016_16-22-43 (CPU: Car	the Line U	tilization)	×						
🧔 🖾	Profile Overview Modules									
✓ ▲ AMDTClassicMatMul Profile Mode (CPU: > ♂ CPU: Time-based Sampling	Processes		🔟 Display Set	tings	Х		All Data, All Mo	<u>dules</u> Display	by: Proces	sses 🔻
> A CPU: Assess Performance	Process PID Cache Line	Útilizatic	Columns		_	viction Acces	ses/L1 Eviction	L1 Evictions	Accesses	s Bytes
CPU: Instruction-based Sampling CPU: Cache Line Utilization	conhost.exe 4412 8.67%		All Data		•	1		69	69	383
 CPO: Cache Line Offizzation Apr-07-2016_16-22-43 	AMDTClassicMatMul.exe 8824 6.29%	Cache Line U			1.06378758 2,524 2,		2,685	10,756		
Overview		🗸 Line Bounda	ary Crossings							
🔞 Modules	<	☑ Bytes/L1Ev	iction						>	
§ Functions		Accesses/L1	1 Eviction							
	Modules filtered by selected processes	L1 Evictions					5.0000000000000000000000000000000000000			
		ache Lin	Accesses			Bytes/L1 Evictio	n Accesses/L	1 Eviction L1	Evictions	Acc ^
	 ⁶⁴ NTFS.sys Not Loaded 12,50% ⁶⁴ KernelBase.dll Not Loaded 12,50% ⁶⁴ msvcrt.dll Not Loaded 12,50% 		Bytes Acces	Bytes Accessed			1	1		1
							1	1		1
	* msvcr.din Not Loaded 1		General				1	5		5
	SRTSP64.SYS Not Loaded 12,50%		Display system modules			1	2		2	
	👫 ntoskrnl.exe Not Loaded 9		Display system modules Show percentages		.07499981	1	40	l.	40	
	* win32kfull.sys Not Loaded 8		CPU Cores	nayes		.16666651	1	12		12
	KernelBase.dll Not Loaded 6	46% 28%				3.73333359 .02132034	3.466666670 1.00533009	15		52 2,45
	MUTClassicillatividi.exe Loaded o		✓ All		^	.02152054	1.00000009	2,4	439	2,4:
Properties 🗗 🛪	sysfer.dll Not Loaded 6		Core 0	Core 1			1	1		1
CPU: Profile Session	📓 ucrtbase.dll 🛛 Not Loaded 6	25%	Core 2	Core 3		2.91666698	3.22916675	48		155 🗸
Profile Type Cache Line Utilization	<		Core 4	Core 5	v					>
Session Name Apr-07-2016_16-22-43			Display data	a per:						
Executable Path: C:\ProgramData\AMD\CodeXL	Cache Line Utilization Notes		Core	, peri						
\Examples \ClassicMatMul \Release \AMDTClassicMatMul.exe										
Working Directory: C:\ProgramData\AMD\CodeXL				OK Ca	ncel					
Profile Scope: Single Application		-			nour					5.00 S
Profile Start Time Apr-07-2016_16-22-43	Modules with a low cache line utilization usual	y indicate	performance bot	tlenecks. Sort the	table acc	ording to a specif	c metric to highlig	ght potential bo	ttleneck mo	dules

© 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Cache Line Utilization (CLU) > Analyzing the CLU profile data > Functions view

Functions view

- 1. To navigate to Functions view, do any of the following:
 - In CodeXL Explorer pane, double click on **Functions** node in the profile session tree entry created for this profile run. The **Functions** view is displayed with function entries from all the profiled processes and the load modules.
 - In **Overview** page, right click on **5 Hottest Modules** table entry and select **Display in Functions View**. The **Functions** view is displayed with function entries from the selected load module.
 - In **Overview** page, right click on **5 Hottest Functions** table entry and select **Display in Functions View**. The **Functions** view is displayed with function entries from all the profiled processes and the load modules.
 - In **Modules** view, select a Process from the **Processes** table, and select **Display in Functions View**. This displays the functions from the selected process and all its load modules.
 - In **Modules** view, select a Module from the **Modules filtered by selected processes** table, and select **Display in Functions View**. This displays the functions from the selected load module.

AMDTClassicMatMul - CodeXL Profile Mode (C	PU: Cache Line Utilization)		- 0	Х
File Edit View Debug Profile Frame Anal	ysis <u>A</u> nalyze <u>T</u> ools <u>W</u> indow <u>H</u> elp			
🏷 🔿 🕶 🖉 🕨 Local 🔻 🖩 🔳				
CodeXL Explorer 🗗 🗙	Apr-07-2016_16-22-43 (CPU: Ca	che Line Utilization) 🛛 🗵		
	🙆 Profile Overview 🔞 Modules 🗵 🔒	Functions 🗵		
 ✓ ▲ AMDTClassicMatMul Profile Mode (CPU: > ♂ CPU: Time-based Sampling 	Functions	Display: <u>All Data, All Modules</u>	17 modules shown, 0 modules hidden Process All Processes	¥
A CPU: Assess Performance	Function	Module	Cache tope Utilization Line Boundary Crossings Bytes/L1 Eviction	^
 CPU: Instruction-based Sampling CPU: Cache Line Utilization 	KeSynchronizeExecution ExplnterlockedPopEntrySList	CodeXL Modules Filter		Х
 Apr-07-2016_16-22-43 Overview 	Ref NTFS.sysl0xf800fda9e2be	Modules of System-wide Profiling(2	Processes):	
Modules	4 EnumResourceNamesExW 4 memset	Modules	Path	^
Functions Properties	 RtlActivateActivationContextUnsafeFast memset RGNMEMOBJ::vPushThreadGuardedObj DEC_SHARE_REF_CNT EnterCrit ThreadUnlock1 HmgLockAllOwners EngUnmapFontFileFD SRTSP64.SYSI0:r800fe4b2730 SRTSP64.SYSI0:r800fe4t4de MmUnmapLockedPages RtlSetBitsEx FsRtlAcquireHeaderMutex 	Image: Second	C:\ProgramData\Symantec\Symantec Endpoint Protection\12.1.6318 C:\ProgramData\Symantec\Symantec Endpoint Protection\12.1.6318 C:\Windows\System32\ C	~
CPU: Profile Session			OK Cancel	
Profile Type Cache Line Utilization	KeGetCurrentIrql			
Session Name Apr-07-2016_16-22-43	<)	>
Executable Path: C:\ProgramData\AMD\CodeXL \Examples\ClassicMatMul\Release \AMDTClassicMatMul.exe Working Directory: C:\ProgramData\AMD\CodeXL	Cache Line Utilization Notes			
\Examples \ClassicMatMul \Release				
Profile Scope: Single Application Profile Start Time Apr-07-2016_16-22-43		sually indicate performance bottleneck	is. Sort the table according to a specific metric to highlight potential bottleneck fun	nctic
Ready				

© 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Analysis with Cache Line Utilization (CLU) > Analyzing the CLU profile data > Source Code view

Source Code view

- 1. To navigate to Source view, do any of the following:
 - In Overview page, right click on 5 Hottest Functions table entry and select Open Source Code. This displays the Source Code view for the selected function.
 - In Functions view, select a function from the Functions table, right click and select Open Source Code from the context menu. This displays the Source Code view for the selected function.
- 2. To view the sources for the some other function, click on the **Function** combo box and select the function for which you want to view the source code.

AMDTClassicMatMul - CodeXL Profile Mode (CP	U: Cache Line Utilization) —	o x
File Edit View Debug Profile Frame Analys	is <u>A</u> nalyze <u>T</u> ools <u>W</u> indow <u>H</u> elp	
🖏 讨 🕶 🔟 🖉 🕨 Local 💌 📗		
CodeXL Explorer 🗗 🗙	III Apr-07-2016_16-22-43 (CPU: Cache Line Utilization)	
a 🔊 🔟	😟 Profile Overview 🛞 Modules 🔟 👔 Functions 🗵 📴Release \AMDTClassidMatMul.exe - Source/Disassembly 🗵	
AMDTClassicMatMul Profile Mode (CPU: O CPU: Time-based Sampling A CPU: Assess Performance O CPU: Instruction-based Sampling CPU: Instruction-based Sampling OU: CPU: Instruction-based Sampling OU: CPU: Instruction-based Sampling OU: Sampling OU: Instruction-based Sampling OU: Instruction OU: In	c: programdata \amd\codexl examples\classicmatmul\amdtclassicmatmul.cpp Function: [0x1510f0 - 0x15120e] : classic_multiply_matrices(void) Display: <u>All Data, All Modules</u> Cache Line Utilization Peri	centage ▼ e Bytes ^
	> 104 0x15120a }	
Properties 🗗 🗙	105	
CPU: Profile Session	106 void improved_multiply_matrices()	
Profile Type Cache Line Utilization	107 {	> ^V
Session Name Apr-07-2016_16-22-43	Cache Line Utilization Notes	
Executable Path: C:\ProgramData\AMD\CodeXL \Examples\ClassidMatMul\Release \AMDTClassidMatMul.exe	No cache line utilization data available for this selection	
Working Directory: C:\ProgramData\AMD\CodeXL \Examples\ClassicMatMul\Release		
Profile Scope: Single Application Profile Start Time Apr-07-2016_16-22-43	() Select a function to find source/disassembly lines with low cache line utilization	
Ready		

© 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Settings for C++ Inline Functions > Microsoft Visual Studio

Microsoft Visual Studio

Prior to Visual Studio 2013 Update 3, inline function information was not preserved in the PDB file. For CodeXL, to extract inline information from PDB file, the target application must be compiled with Visual Studio 2013 Update 3 or later. Use the below Visual Studio Configuration Properties if you want to enable inline function details in CodeXL CPU Profiler.

Set Configuration Properties > C/C++ > General > Debug Information Format to /Zi. Without this flag, PDB file with debug info will not be generated.

AMDTClassicMatMul Property Pages			? X
<u>C</u> onfiguration: Debug	✓ Platform: Active(Win32)		✓ Configuration Manager
 ✓ Configuration Properties General Debugging VC++ Directories ✓ C/C++ General Optimization Preprocessor Code Generation Language Precompiled Headers Output Files Browse Information Advanced All Options Command Line > Linker > Manifest Tool > XML Document Generator > Browse Information > Build Events > Custom Build Step > Code Analysis 	Additional Include Directories Additional #using Directories Debug Information Format Common Language RunTime Support Consume Windows Runtime Extension Suppress Startup Banner Warning Level Treat Warnings As Errors Warning Version SDL checks Multi-processor Compilation	Program Database (/Zi) Yes (/nologo) Level3 (/W3) No (/WX-)	
< >	Debug Information Format Specifies the type of debugging information (/Z7, /Zi, /Zl)	generated by the compiler. This requ	ires compatible linker settings.
		ОК	Cancel <u>Apply</u>

Add /Zo to Configuration Properties > C/C++ > Command Line > Additional Options. Without this flag, additional debug info related to inline functions will not be generated. /Zo only recognized by VS2013 update 3 or later. The /Zo option is enabled by default in Visual Studio 2015 when you specify debugging information with /Zi or /Z7, hence this step is optional for VS2015.

AMDTClassicMat	tMul Property Pages			? X
<u>C</u> onfiguration:	Debug	✓ <u>P</u> latfor	n: Active(Win32)	✓ Configuration Manager
General Debugg VC++ [✓ C/C++ Gen Opt Prej Coo Lan Pred Out Bro Adv All (<u>Cor</u> ▷ Linker ▷ Manife	ging Directories heral imization processor de Generation guage compiled Headers sput Files wse Information vanced Options mmand Line	"_DEBUG" /D "_CO	Zc:wchar_t /Zi /Gm /Od /Fd"Debug\vc140.pdb" /Zc:inline /fp ISOLE" /D "_UNICODE" /D "UNICODE" /errorReport:prompt /V ug\" /EHsc /nologo /Fo"Debug\" /Fp"Debug\AMDTClassicMa	VX- /Zc:forScope /RTC1 /Gd
▷ Browse ▷ Build Ev	Information			~
	n Build Step	A <u>d</u> ditional Options	Inherit fi	rom parent or project defaults 🗹
▷ Code A		/Zo		^
<	>		ОК	Cancel <u>Apply</u>

Enable Optimization by setting any one of: /O1, /O2, /Ox (Configuration Properties > C/C++ > Optimization > Optimization). Setting it to /Od will stop all optimizations including function inline.

Enable Function inlining by setting one of: /Ob1, /Ob2 (Configuration Properties > C/C++ > Optimization > Inline Function Expansion). Setting it to /Ob0 will stop function inline.

•

AMDTClassicMa	tMul Property Pages						?	Х
<u>Configuration</u> :	Debug	✓ Platform:	Active(Win32)		~	Configuration	Manag	er
Genera Debugy VC++ I ✓ C/C++ Ger Op Pre Coo Lan Pre Out Bro Adv All Cor È Linker ▷ Manife ▷ XML Do B rowse ▷ Build E	ging Directories heral timization processor de Generation iguage compiled Headers tput Files wse Information vanced Options mmand Line st Tool ocument Generator e Information	Optimization Inline Function Ex Enable Intrinsic Fu Favor Size Or Spee Omit Frame Point Enable Fiber-Safe Whole Program O	inctions ed ers Optimizations	Maximize Speed (/O2) Only_inline (/Ob1) No No (/Oy-) No No				
Code A	Inalysis	Inline Function Expan Select the level of inlin		n for the build. (/Ob1, /Ob2)	ОК	Cancel	Ар	ply

 On Visual Studio 2015, enable "Whole Program Optimization" (Configuration Properties > C/C++ > Optimization > Whole Program Optimization). This setting is not required for VS2013.

Inlined Classic Stress Property Pages			? 💌
Configuration: Debug	<u>P</u> latform:	Active(Win32)	▼ Configuration Manager
 Configuration Properties General Debugging VC++ Directories C/C++ General Optimization Preprocessor Code Generation Language Precompiled Headers Output Files Browse Information Advanced All Options Command Line Linker Manifest Tool XML Document Generator Browse Information Walifest Tool XML Document Generator Browse Information Build Events Custom Build Step Code Analysis 	Optimization Inline Function Ep Enable Intrinsic F Favor Size Or Spe Omit Frame Poin Enable Fiber-Safe Whole Program O Whole Program Opt Enables cross-modul Time Code Generatio	imization ed ters Optimizations Dytimization	ne; requires that linker option 'Link
			OK Cancel <u>A</u> pply

Disable data elimination by setting /OPT:NOREF (**Configuration Properties > Linker > Optimization > References**). Setting it to /OPT:REF eliminates unreferenced data/function during linking time. In Release build, by default REF is enabled, due to which inline function info gets removed from PDB file.

Disable COMDAT folding by setting /OPT:NOICF (Configuration Properties > Linker > Optimization > Enable COMDAT Folding). Setting it to /OPT:ICF combines identical functions. In Release build, by default ICF is enabled, due to which identical inline functions info gets removed from PDB file.

•

AMDTClassicMatMul Property Pages						?	Х
<u>C</u> onfiguration: Debug	✓ Platform:	Active(Win32)		~	C <u>o</u> nfiguration	Manag	er
 Configuration Properties General Debugging VC++ Directories C/C++ Linker General Input Manifest File Debugging System Optimization Embedded IDL Windows Metadata Advanced All Options Command Line Manifest Tool XML Document Generator Browse Information Build Events Custom Build Step Code Analysis 	References Enable COMDAT f Function Order Profile Guided Dat Link Time Code G	tabase	No (/OPT:NOREF) No (/OPT:NOICF) \$(OutDir)\$(TargetNam	ne).pgd			
< >	Enable COMDAT Fold Use /OPT:ICF[=iteration		entical COMDAT folding.				
				ОК	Cancel	Ap	ply

© 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Settings for C++ Inline Functions > GCC/G++ Compiler

GCC/G++ Compiler

- Enable debug info using "-g" or "-gdwarf-version". Without this option, target application binary will not include any debugging information.
- Enable optimization using any one of: -O, -O1, -O2, or -O3.
 Without this option, compiler will disable any function inlining.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Settings for C++ Inline Functions > Session Views

Session Views

Following screenshots for Overview, Call Graph view and Source Code view shows the inline functions for a sample target application.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Settings for C++ Inline Functions > Session Views > Overview page

Overview page

	•								
CodeXL Explorer	ā ×	Ó A	pr-21-2015_17-51	-37 (CPU: Time-	-based Sampling)	×			
(AMDZI	0 Profile Overview							
 SingleInline-d Profil OPU: Time-based A (a) Ann 21, 2015 	d Sampling	5 Hottest Functions				Display: System Modules Hidden Hotspot Indicator Time			
 Ø Apr-21-2015_ Ø Apr-21-2015 		Function		Samples	% of Hotspot Samples	Module			
 Overview 		[inlined] multiply_matrices(void		59,349	99.77%	SingleInline-d.exe			
🖗 Modules		initialize_matrice	s(void)	132	0.22%	SingleInline-d.exe			
📱 Call Grap	h	🕍 main		2	0.00%	SingleInline-d.exe			
§ Functions	5	⁶⁴ EraserUtilReboot			0.00%	EraserUtilRebootDrv.sys			
	BHDrvx64.sys!+0:	xe//c0	1	0.00%	BHDrvn64.sys				
		5 Hottest Modules							
		Module	Samples	% of H	otspot Samples				
Properties	₽ X	₽ ⁶⁴ BHDrvx64.sys	1	0.00%					
CPU: Profile Session	15 17 51 07								
Profile Name Apr-21-201 Profile Type Time-based	15_17-51-37	CPU Affinity: Oxf			Profile Scope:	Tarda Analia Era			
rionic type time-based									
Executable Path: C:\AMD\Single	eInline\Debug	Call Stack Sampling	True		Call Stack 1110	rmation: Scope - User space, Depth - 128, Frequency - per 1 samples, FPO - Off			
Executable Path: C: \AMD\Single \SingleInline-d.exe		Call Stack Sampling	: True		Call Stack 100	rmation: Scope - User space, Depth - 128, Frequency - per 1 samples, FPO - Off			
Executable Path: C: \AMD\Single \SingleInline-d.exe Working Directory: C: \AMD\Sing Profile Scope: Single Application	gleInline\Debug	Call Stack Sampling Profile Details	: True		Call Stack 100	rmation: Scope - User space, Depth - 128, Frequency - per 1 samples, FPO - Off			
Executable Path: C: \AMD\Single \SingleInline-d.exe Working Directory: C: \AMD\Sing Profile Scope: Single Application Profile Start Time Apr-21-201	gleInline\Debug 1 15_17-51-37			pling	Can Stack Into				
Executable Path: C: \AMD\Single \SingleInline-d.exe Working Directory: C: \AMD\Sing Profile Scope: Single Application Profile Start Time Apr-21-201 Profile End Time Apr-21-201	gleInline Debug 1 15_17-51-37 15_17-52-39	Profile Details Profile Session Type Profile Start Time: A	2: Time-based Sam pr-21-2015_17-5	1-37					
Executable Path: C: \AMD\Single \SingleInline-d.exe Working Directory: C: \AMD\Sing Profile Scope: Single Application Profile Start Time Apr-21-201 Profile End Time Apr-21-201	gleInline Debug 1 15_17-51-37 15_17-52-39	Profile Details Profile Session Type	2: Time-based Sam pr-21-2015_17-5	1-37	Monitored Eve				
Executable Path: C: \AMD\Single \SingleInline-d.exe Working Directory: C: \AMD\Sing Profile Scope: Single Application Profile Start Time Apr-21-201 Profile End Time Apr-21-201 CPU Details: Family 0x15, Model	gleInline Debug 1 15_17-51-37 15_17-52-39	Profile Details Profile Session Type Profile Start Time: A	r: Time-based Sam pr-21-2015_17-5 21-2015_17-52-	1-37	Monitored Eve				
Executable Path: C: \AMD\Single \SingleInline-d.exe Working Directory: C: \AMD\Sing Profile Scope: Single Application Profile Start Time Apr-21-201 Profile End Time Apr-21-201 CPU Details: Family 0x15, Model Profile Duration: 1:01	gleInline Debug 1 15_17-51-37 15_17-52-39	Profile Details Profile Session Type Profile Start Time: A Profile End Time: Ap	e: Time-based Sam pr-21-2015_17-5 -21-2015_17-52- 1	1-37 39	Monitored Eve				
Executable Path: C: \AMD\Single \SingleInline-d.exe Working Directory: C: \AMD\Single Profile Scope: Single Application Profile Start Time Apr-21-201 Profile End Time Apr-21-201 CPU Details: Family 0x15, Model Profile Duration: 1:01 CPU Affinity: 0xf	gleInline Debug 15_17-51-37 15_17-52-39 1 16, 4 core(s) User space, Depth - 128,	Profile Details Profile Session Type Profile Start Time: A Profile End Time: Ap Profile Duration: 1:0	e: Time-based Sam pr-21-2015_17-5 r-21-2015_17-52- 1 c15, Model 16, 4 c	1-37 39	Monitored Eve				
Executable Path: C: \AMD\Single \SingleInline-d.exe Working Directory: C: \AMD\Sing Profile Scope: Single Application Profile Start Time Apr-21-201 Profile End Time Apr-21-201 CPU Details: Family 0x15, Model Profile Duration: 1:01 CPU Affinity: 0xf Call Stack Information: Scope -	gleInline Debug 15_17-51-37 15_17-52-39 1 16, 4 core(s) User space, Depth - 128,	Profile Details Profile Session Type Profile Start Time: A Profile End Time: Ap Profile Duration: 1:0 CPU Details: Family 0:	e: Time-based Sam pr-21-2015_17-5 r-21-2015_17-52- 1 c15, Model 16, 4 c	1-37 39	Monitored Eve				

Inline function "multiply_matrices(void)" is shown in the 5 Hottest Functions table. $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Settings for C++ Inline Functions > Session Views > Call Graph View

Call Graph View

odeXL Explorer	₽×	Ò	Apr-21-2015_17-5	L-37 (CPU: Time-ba	ased Sampling)	×					
	AMDZI	Profile Overview	📕 Call Graph	X							
 MingleInline-d Profile Mode OPU: Time-based Sampl Apr-21-2015_17-42-3 	ling	Functions				Display: <u>System M</u>	Modules Hidden	Process: Si	ingleInline	e-d.exe(8224) 🔻 M	Monitored event:
4 d Apr-21-2015_17-42-5		Function (29 funct	ions, 5 shown)	Self Samples	Deep Samples	% of Deep Samp	ples No. of Paths	Source File	١	Module	
0 Overview		mainCRTStartu	ib		59711	100.00%	20	crtexe.c(456)		SingleInline-d.exe	
🛞 Modules		_tmainCRTStar	tup		59711	100.00%	20	crtexe.c(473)		SingleInline-d.exe	
📕 Call Graph		main		2	59351	99.40%	2			SingleInline-d.exe	
👔 Functions		(inlined) multip initialize_matric	oly_matrices(void)	59349 132	59349 355	99.39% 0.59%	1 16	-		SingleInline-d.exe SingleInline-d.exe	
PU: Profile Session	₿ × 1-37	Immediate Parents an		on: [inlined] mul		roid)	Self + children	S	Samples	% of samples	Module
CPU: Profile Session Profile Name Apr-21-2015_17-5: Profile Type Time-based Sampin Executable Path: C: \AMD\SingleInline\D SingleInline-d.exe Working Directory: C: \AMD\SingleInline	1-37 ng Jebug						Self + children [self]			% of samples 100.00%	Module SingleInline-d.exe
CPU: Profile Session Profile Name Apr-21-2015_17-53 Profile Type Time-based Samplin Executable Path: C:\AMD\SingleInline\D SingleInline-d.exe Norking Directory: C:\AMD\SingleInline\ Profile Scope: Single Application	1-37 ng Debug VDebug	Parents	Sampl	es % of samples	Module			59	59349		
CPU: Profile Session Profile Name Apr-21-2015_17-5: Profile Type Time-based Samplin SingleInline Value C: \AMD\SingleInline \D SingleInline-d.exe Norking Directory: C: \AMD\SingleInline\P Yrofile Scope: Single Application Profile Start Time	1-37 ng Debug VPebug 1-37	Parents main	Sampl	es % of samples 100.00%	Module SingleInline-d.e	xe	[self]	59 In selection path	9349		
PU: Profile Session Profile Name Apr-21-2015_17-51 Profile Type Time-based Samplin Executable Path: C: (AMD) (SingleInline) SingleInline-d.exe Vorking Directory: C: (AMD) (SingleInline) rofile Scope: Single Application rofile Start Time Apr-21-2015_17-52 rofile End Time Apr-21-2015_17-52	1-37 ng Debug 1-37 2-39	Parents main Paths Function / mainCRTStartu	Sampi 59349	es % of samples 100.00%	Module SingleInline-d.e	xe ream samples १	[self]	59 In selection path amples Modu	i9349 ule	100.00%	
	1-37 ng Debug 1-37 2-39	Parents main Paths Function	Sampi 59349	es % of samples 100.00%	Module SingleInline-d.e amples Downst 59349 59349	xe ream samples 9 1	[self] Show Call Grap % of Downstream s 100.00%	59 h selection path amples Modu Single Single	19349 ule eInline-d. eInline-d.	100.00%	
CPU: Profile Session Profile Name Apr-21-2015_17-51 Profile Type Time-based Samplin SingleInline-Date C:\AMD\SingleInline\D SingleInline-d.exe Working Directory: C:\AMD\SingleInline\D Yrofile Scope: Single Application Yrofile Start Time Yrofile Start Time Apr-21-2015_17-51 Yrofile End Time Apr-21-2015_17-55 Yrofile End Time Apr-21-2015_17-55 YPU Details: Family 0x15, Model 16, 4 cd	1-37 ng Debug 1-37 2-39	Parents main Paths Function MainCRTStartu TmainCRT Main	Sampi 59349	es % of samples 100.00% Self s	Module SingleInline-d.e amples Downst 59349 59349 59349	xe ream samples 9 1 1 1	[self] Show Call Grap % of Downstream s 100.00%	54 h selection path amples Modu Single Single	i9349 ule eInline-d.	100.00%	

Inline function "multiply_matrices(void)" is shown in the Functions table and Paths table.

 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Settings for C++ Inline Functions > Session Views > Source Code View

Source Code View

🗵 SingleInline-d - CodeXL Profile Mode (CPU: Time	e-based Sar	mpling)					- • ×
<u>File E</u> dit <u>V</u> iew <u>D</u> ebug <u>P</u> rofile <u>A</u> nalyze <u>T</u> o	ools <u>W</u> in	dow <u>H</u> elp					
🏷 👩 🛛 🕨 🕨 Local 🔻 📗 🔛							
CodeXL Explorer 🗗 🗙	Ò		Apr-21-2015_17-51-37 (CPU: Time-based Sampling)	×			
AMDZI	O Profi	ile Overview	🚱line\Debug\SingleInline-d.exe - Source/Disassembly 🛛				
4 🎪 SingleInline-d Profile Mode (CPU: Time	c:\amd\s	inaleinline\sina	Jeinline \singleinline.cpp				
4 👌 CPU: Time-based Sampling			3 - 0x9311cb] : [inlined] multiply_matrices(void) V Display: All D	ata. Svetem Modulae Hiddan		Hot	spot Indicator Timer 🔻
▷ ġ Apr-21-2015_17-42-30		III [[0X301000	oroxoorreal, filmileal indiaply_indiaeles(void)	ata, system Houses Houer		100	
▲ 🙆 Apr-21-2015_17-51-37	Line	Address	Source Code	Code Bytes	Hotspot Samples	% of Hotspot Samples	Timer
 Overview Modules 	10		matrix_p(j)j = (noarjiano() / NAINO_MAA; matrix_r(j)[j] = 0.0;				
Call Graph	20						
Functions	21		}				
4 🔂 Source Code	22		}				
G SingleInline-d.exe	23						
	24		_forceinline void multiply_matrices()				
	25		{				
	26		// Multiply the two matrices				
	▶ 27	0x9310b8	for (int i = 0; i < ROWS; i++) {				
Properties 🗗 🗙	▷ 28	0x9310c0	for (int j = 0; j < COLUMNS; j++) {				
CPU: Profile Session	≥ 29	0x9310d6	float sum = 0.0;		5	0.01%	5
Profile Name Apr-21-2015_17-51-37	30 ▷ 31	0x9310e2	for (int k = 0; k < COLUMNS; k++) { sum = sum + matrix_a[i][k] * matrix_b[k][j];		59,203	99.75%	59,203
Profile Type Time-based Sampling	32	0/931062	sum = sum + marix_a(i)(k) - marix_b(k)(j); }		39,205	99.1076	59,205
Executable Path: C:\AMD\SingleInline\Debug	▷ 33	0x93119a	, matrix_r[i][j] = sum;		141	0.24%	141
SingleInline-d.exe	34		}			012470	
Working Directory: C:\AMD\SingleInline\Debug	35		}				
Profile Scope: Single Application	36		}				Ξ
Profile Start Time Apr-21-2015_17-51-37	37						
Profile End Time Apr-21-2015_17-52-39	38		int main(int argc, char* argv[])				
CPU Details: Family 0x15, Model 16, 4 core(s)	39		{				
Profile Duration: 1:01	40		initialize_matrices();				
CPU Affinity: 0xf	41		multiply_matrices();				
Call Stack Information: Scope - User space, Depth - 128,	42		return(0);				
Frequency - per 1 samples, FPO - Off	44		return(u);				_
			1				
	🚺 Se	elect a function	n from the drop-down list to display its source/disassembly lines with	sample counts			
Ready							đ

Samples are attributed to corresponding source lines for Inline function "multiply_matrices(void)".

 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Settings for CPU Profiler to run on Virtual Machine

Settings for CPU Profiler to run on Virtual Machine

CPU Profiler supports profiling on guest OS running on VMware workstation.

 $\ensuremath{\textcircled{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > Settings for CPU Profiler to run on Virtual Machine > VMware Workstation

VMware Workstation

CPU Profiler supports TBP and EBP on guest OS running on VMware Workstation 11.0 or later. Recent AMD Carrizo processor is not yet supported by VMware Workstation 11.1.x.

To run TBP within guest OS, no additional configuration needed in host OS or guest OS.

To run EBP within guest OS, please ensure the following settings are done:

- Enable Virtualization or SVM (AMD-V) in BIOS settings before booting the host OS.
- Enable AMD-V in guest OS VM settings. Edit virtual machine settings > Hardware > Processors> Virtualization engine > Enable "Virtualize AMD-V/RVI"
- Enable vPMC in guest OS VM settings. Edit virtual machine settings > Hardware > Processors> Virtualization engine > Enable "Virtualize CPU performance counters"

Known Issues on Windows 7 Host OS:

- When CPU Profiler EBP is running on Windows 7 host OS and a Linux guest OS is launched, crash is observed on Windows 7 due to VMware driver.
- If EBP is performed on Windows 7 host OS and EBP is performed on Linux guest OS simultaneously, then crash is observed on Windows 7 due to VMware driver.

These scenarios work fine when host OS is Windows 8, 8.1 and 10.

 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > How to use CPU Profiler Control APIs

How to use CPU Profiler Control APIs

CPU Profile control APIs provide the option to restrict the profiling to a specific part of the code. Follow the below steps to call, compile and invoke these APIs from your target program.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > How to use CPU Profiler Control APIs > Required File Paths

Required File Paths

All the paths mentioned below are with respect to CodeXL installation directory (CODEXL-INSTALL-DIR).

API Documentation: <CODEXL-INSTALL-DIR>\SDK\AMDTActivityLogger\doc\AMDTActivityLogger.pdf (on Windows platforms) OR <CODEXL-INSTALL-DIR>/SDK/AMDTActivityLogger/doc/AMDTActivityLogger.pdf (on Linux platforms) Header File: <CODEXL-INSTALL-DIR>\SDK\AMDTActivityLogger\include\AMDTActivityLogger.h (on Windows platforms) OR <CODEXL-INSTALL-DIR>/SDK/AMDTActivityLogger/include/AMDTActivityLogger.h (on Linux platforms) LIB Files (on Windows platforms): <CODEXL-INSTALL-DIR>\SDK\AMDTActivityLogger\lib\x64\AMDTActivityLoggerx64.lib <CODEXL-INSTALL-DIR>\SDK\AMDTActivityLogger\lib\x86\AMDTActivityLogger.lib **DLL Files (on Windows platforms):** <CODEXL-INSTALL-DIR>\SDK\AMDTActivityLogger\bin\x64\AMDTActivityLoggerx64.dll <CODEXL-INSTALL-DIR>\SDK\AMDTActivityLogger\bin\x86\AMDTActivityLogger.dl

Shared Library (.so) Files (on Linux platforms): <CODEXL-INSTALL-

DIR>/SDK/AMDTActivityLogger/bin/x86_64/libAMDTActivityLog <CODEXL-INSTALL-

DIR>/SDK/AMDTActivityLogger/bin/x86/libAMDTActivityLogger

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > How to use CPU Profiler Control APIs > Calling the APIs

Calling the APIs

Include the header file AMDTActivityLogger.h, and call the resume and pause APIs within the code. The code encapsulated within resume-pause API pair will be profiled by CPU Profiler.

These APIs can be called multiple times to profile different parts of the code.

These APIs can be spread across multiple functions, i.e. resume called from one function and stop called from another function. These APIs can be spread across threads, i.e. resume called from one thread and stop called from another thread of the same target application.

In the below example, the CPU Profiler is restricted to the execution of multiply_matrices() function.

```
#include <AMDTActivityLogger.h>
int main(int argc, char* argv[])
{
  // Initialize the matrices
  printf("Initializing matrices\n");
  initialize_matrices();
  // Multiply the matrices
  printf("Multiplying matrices\n");
  // Resume the CPU profiler
  // amdtResumeProfiling(AMDT CPU PROFILING);
  amdtResumeProfilingEx();
  multiply_matrices();
  // Stop the CPU Profiler
  // amdtStopProfiling(AMDT_CPU_PROFILING);
  amdtStopProfilingEx();
  return 0:
}
```

 $\ensuremath{\textcircled{\sc 0}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > How to use CPU Profiler Control APIs > Compiling the target application

Compiling the target application

To compile the application on Microsoft Visual Studio, update the configuration properties to include the path of header file, DLL file, LIB file of AMDTActivityLogger.

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

\$ g++ -std=c++0x <source-file.cpp> -L<CODEXL-INSTALL-DIR>/SDK/AMDTActivityLogger/bin/x86_64/ -L<CODEXL-INSTALL-DIR> -I<CODEXL-INSTALL-DIR>/SDK/AMDTActivityLogger/include -lAMDTBaseTools -lAMDTOSWrappers lAMDTActivityLogger -lrt -ldl

You may choose to use 2011 C++ standard or newer C++ standard while compiling C++ target application with profile control APIs.

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

\$ gcc <source-file.c> -L<CODEXL-INSTALL-DIR>/SDK/AMDTActivityLogger/bin/x86_64/ -L<CODEXL-INSTALL-DIR> -I<CODEXL-INSTALL-DIR>/SDK/AMDTActivityLogger/include lAMDTBaseTools -lAMDTOSWrappers -lAMDTActivityLogger -lrt

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Tutorials > CPU Profiler Tutorial > How to use CPU Profiler Control APIs > Profiling with Control APIs

Profiling with Control APIs

After the compiling the target application, <u>create a project in</u> <u>CodeXL</u>, <u>set the general profile session options</u>, <u>set the CPU</u> <u>profile session options</u>. While setting the CPU profile session options, in step 4, in the **Data Collection Schedule** section, select **Start profile with data collection paused**.

Once all the settings done, start the CPU profiling. The CPU Profiler will be in pause state and target application execution begins. When the resume API gets called from target application, CPU Profile starts profiling till stop API gets called from target application. As soon as stop API is called in target application, CPU Profiler stops profiling and waits for next control API call.

To profile from CLI, option '-s 0' should be used to start the profiler in pause state.

Sample command on Windows platforms:

> CodeXLCpuProfiler.exe collect -m tbp -s 0 -o c:\Temp\cpuprof-tbp <target-application.exe>

Sample command on Linux platforms:

\$ CodeXLCpuProfiler collect -m tbp -s 0 -o /tmp/cpuprof-tbp <target-application>

^{© 2015} Advanced Micro Devices. All rights reserved.





Help > Tutorials > GPU Profiler Tutorial

GPU Profiler Tutorial

The CodeXL GPU Profiler is a performance analysis tool that gathers data from the OpenCL run-time and AMD RadeonTM GPUs during the execution of an OpenCL application.

Use this information to discover bottlenecks in an application and to find ways to optimize the application's performance for AMD platforms. There are two modes of operation supported by the Profiler.

Collecting OpenCL Application Trace

Collecting OpenCL GPU Kernel Performance Counters

© 2015 Advanced Micro Devices. All rights reserved.



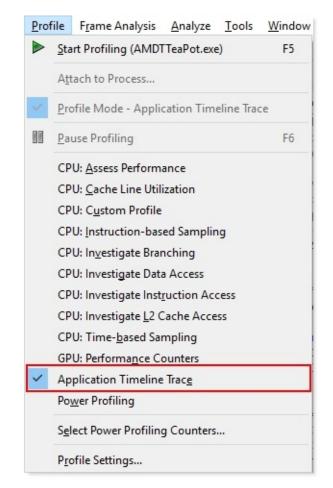


Help > Tutorials > GPU Profiler Tutorial > Collecting OpenCL Application Trace

Collecting OpenCL Application Trace

Run an Application Trace GPU Profile session

- 1. Open or create a CodeXL project.
- 2. Select the Application Timeline Trace profile mode



- **3**. Click the (Start Profiling) toolbar button to start profiling.
- 4. Stop the profiled application.

When the profiled application's execution is over, CodeXL displays the session.

🖾 CodeXL Teapot - CodeXL Profile Mode (Applicatic	n Tinnalina Traca)					- [J X
File Edit View Debug Profile Frame Analysis	<u>Analyze Iools W</u> indow <u>H</u> elp						
🏷 🔿 🗖 🖳 🕨 🖉							
					pos	X V A Mate	th <u>C</u> ase X
Apr-10-2016_11-54-5	7 (Application Timeline Trace)	×					
Application Timeline Trace							
119 1198.310	<mark>19.402</mark> 199.546 1200.782 1202.018	1203.254 1204.490) 1205.726	1206.962 1208.19	1209.435 12	210.671 1211.907 1	1213.143
OpenCL eueWrite	dEnqueueNDRangeKernel ueNDRangeue	ueNDRangeK	dEnqueueNDRangeKe	rnel ueAcquireGL	dFinish		٨
⊡ OpenCL							
⊡ Context 0 (0x0EA59990)							
□ Queue 1 - Hawaii (0x0F40D970)							
Data Transfer WRITE_E							
Kernel Execution		ſq					
Other Enqueue Operations							
□ Queue 0 - CPU_Device (0x0F3C3EF0)							
Data Transfer							
Kernel Execution				n			¥
<							>
Host Thread 8900 Summary							
CL API Summary							
							٨
API Name	Cumulative Time(ms)	% of Total Time	# of Calls	Avg Time(ms)	Max Time(ms)	Min Time(ms)	11
clBuildProgram	1080.65627	52.81785	17	63.56801	<u>94.88455</u>	<u>51.64770</u>	- 11
clEnqueueNDRangeKernel	479.32810	23.42750	25465	0.01882	2.18533	<u>0.01076</u>	-11
clFinish	254.33322	12.43072	926	0.27466	<u>1.91253</u>	0.01564	_
clCreateCommandQueue	50.39809	2.46324	2	25.19905	46.63805	3.76004	_
clReleaseEvent	36.26032	1.77225	25465	0.00142	0.09093	0.00098	_
clEnqueueWriteBuffer	28.57213	1.39648	932	0.03066	<u>3.89107</u>	0.01124	_
clCreateContext	26.95000	1.31720	1	26.95000	<u>26.95000</u> 0.42511	26.95000	v
i aufistEartuante	1 71 /W/KK		1 467	n mikuu	1.0.42811	1 CORTAG	

Timeline View

Application Timeline Trace													
	1: 1198.310	199.402 1199.546	1200.782	1202.018	1203.254	1204,490	1205.726	1206.962	1208.198	1209.435	1210.671	1211.907	1213.143
OpenCL	eueWrite W	dEnqueu	eNDRangeKernel	eueNDRangeueue	NDRangek	dEn	queueNDRangeKe	ernel ueAcquire	eGL	dFinish			
⊡ OpenCL													
⊡ Context 0 (0x0EA59990)													
🗆 Queue 1 - Hawaii (0x0F40D970)													
Data Transfer	WRITE_E												
Kernel Execution						Va							
Other Enqueue Operations													
□ Queue 0 - CPU_Device (0x0F3C3EF0)													
Data Transfer													
Kernel Execution													

The Timeline View provides a visual representation of the execution of the application. Along the top of the timeline is the time grid, showing the total elapsed time (in milliseconds) of the application when fully zoomed out. Timing begins when the first OpenCL call is made by the application; it ends when the final OpenCL call is made. Directly below the time grid, each host (OS) thread that made at least one OpenCL call is listed. For each host thread, the OpenCL API calls are plotted along the time grid, showing the start time and duration of each call. Below the host threads, the OpenCL tree shows all contexts and gueues created by the application, along with data transfer operations and kernel execution operations for each queue. We can navigate in the Timeline View by zooming, panning, collapsing/expanding or selecting a region of interest. From the Timeline View, we can also navigate to the corresponding API call in the API Trace View and vice versa. The Timeline View can be useful for debugging your OpenCL application. Some examples are:

You easily can confirm that the high-level structure of your application is correct. By examining the timeline, you can verify that the number of queues and contexts created match your expectations for the application.

You also can confirm that synchronization has been performed properly in the application. For example, if kernel A execution is dependent on a buffer operation and outputs from kernel B execution, then kernel A execution appears after the completion of the buffer execution and kernel B execution in the time grid. It can be hard to find this type of synchronization error using traditional debugging techniques.

You also can confirm that the application has been using the hardware efficiently: the timeline should show that non-dependent kernel executions and data transfer operations occur simultaneously.

Context ID		# of Images	# of Kernel Dispatch - Hawaii		Dispatch -	Total Kernel Time(ms) - CPU_Device	Memory		# of Read	Total Read Time(ms)	Size of Read	# of Write	Total Write Time(ms)	Size of Write	# of Map	Total Map Time(ms)	Size of Map	# of Copy	Total Copy Time(ms)	Size of Copy
0	14	0	25002	603.42622	463	45.68129	1858	27.40173	0	0	0 Byte	932	5.88489	11.18 MB	463	1.05699	14.12 MB	<mark>4</mark> 63	20.45985	1.81 GB
Total	14	0	25002	603.42622	463	45.68129	1858	27.40173	0	0	0 Byte	932	5.88489	11.18 MB	463	1.05699	14.12 MB	463	20.45985	1.81 GB

Summary Pages View

The Summary Pages View shows various statistics for your OpenCL application. It can provide you with a general idea of the location of the application's bottlenecks. It also provides useful information, such as the number of buffers and images created on each context, the most expensive kernel call, etc. The Summary Pages View provides access to the following individual pages:

- API Summary page Shows statistics for all OpenCL API calls made in the application for API hotspot identification.
- Context Summary page Shows the statistics for all the kernel dispatch and data transfer operations for each context. It also shows the number of buffers and images created for each context.
- Kernel Summary page Shows statistics for all the kernels that are created in the application.
- Top 10 Data Transfer Summary page Shows a sorted list of the ten most time-consuming individual data transfer operations.
- Top 10 Kernel Summary page Shows a sorted list of the ten most time-consuming individual kernel execution operations. From these summary pages, it is possible to determine whether the application is bound by kernel execution or data transfer (Context Summary page). If the application is bound by kernel execution, you can determine which device is the bottleneck. From the Kernel Summary page, we can find the name of the kernel with the highest total execution time. Or, from the Top 10 Kernel Summary page, we can find the individual kernel instance with the highest execution time. If the kernel execution on a GPU device is the bottleneck, the GPU performance counters then can be used to investigate the bottleneck inside the kernel. See Collecting OpenCL GPU Kernel Performance Counters for more details. If the application is bound by the data transfers, it is possible to determine the most time-consuming data transfer type (read, write, copy or map) in the application from the Context Summary page. You can investigate whether it is possible to minimize this type of data transfer by modifying the algorithm. With help from the Timeline View, you can investigate whether data transfers have

been executed in the most efficient way (that is: concurrently with a kernel execution).

API Trace View

Call Index	Interface	Parameters	Result	Device Block	Kernel Occupancy	CPU Time	Device Time	^
26170	clReleaseEvent	0x0C5AF678	CL_SUCCESS			0.0015		
26171	clFlush	0x0F40D970	CL_SUCCESS			0.0068		
26172	clEnqueueWriteBuffer	0x0F3C3EF0;0x19576C48;CL_FALSE;0;288;0x0E98E478;0;NULL;NULL	CL_SUCCESS	288.0 Byte WRITE		0.0127	0.0083	
26173	clEnqueueNDRangeKernel	0x0F3C3EF0;0x0F8B5E40;1;NULL;[111];NULL;0;NULL;NULL	CL_SUCCESS	computeIntersect		0.0137	0.0826	
26174	clFlush	0x0F3C3EF0	CL_SUCCESS			0.0039		
26175	clEnqueueAcquireGLObjects	0x0F40D970;1;[0x1986E9C0];0;NULL;NULL	CL_SUCCESS	ACQUIRE GL OBJ		0.0186		1
26176	clEnqueueCopyBufferTolmage	0x0F40D970;0x0F9B29D8;0x1986E9C0;0;[0,0,0];[64,64,64];0;NULL;NULL	CL_SUCCESS	4.0 MB COPY BU		0.0132	0.0434	
26177	clEnqueueReleaseGLObjects	0x0F40D970;1;[0x1986E9C0];0;NULL;NULL	CL_SUCCESS	RELEASE GL OBJ		0.0235		
26178	clEnqueueMapBuffer	0x0F3C3EF0;0x19576748;CL_FALSE;CL_MAP_READ;0;31968;0;NULL;[0x19338F90];[CL_SUCCE	0x194F4000	31.2 KB MAP BUF		0.0147	0.0024	
26179	clFlush	0x0F40D970	CL_SUCCESS			0.0044		
26180	clFlush	0x0F3C3EF0	CL_SUCCESS			0.0044		
26181	clWaitForEvents	1;[0x19338F90]	CL_SUCCESS			0.0420		
26182	clEnqueueUnmapMemObject	0x0F3C3EF0;0x19576748;0x194F4000;0;NULL;NULL	CL_SUCCESS	UNMAP MEM O		0.0254	0.0108	
26183	clFinish	0x0F40D970	CL_SUCCESS			0.5617		
26184	clFinish	0x0F3C3EF0	CL SUCCESS			0.0225		۷

The API Trace View lists all the OpenCL API calls made by the application. Each host thread that makes at least one OpenCL call is listed in a separate tab. Each tab contains a list of all the API calls made by that particular thread. For each call, the list displays the index of the call (representing execution order), the name of the API function, a semi-colon delimited list of parameters passed to the function and the value returned by the function. When displaying parameters, the Profiler tries to dereference pointers and decode enumeration values to give as much information as possible about the data being passed in or returned from the function. Double-clicking an item in the API Trace View displays and zooms into that API call in the Host Thread row in the Timeline View.

This view lets you analyze and debug the input parameters and

output results for each API call. For example, you easily can check that all the API calls return CL_SUCCESS, or that all the buffers are created with the correct flags. This view also lets you identify redundant API calls.

© 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > GPU Profiler Tutorial > Collecting OpenCL GPU Kernel Performance Counters

Collecting OpenCL GPU Kernel Performance Counters

Run a Performance Counters GPU Profile session

- 1. Open or create a CodeXL project.
- 2. Select the GPU: Performance Counters profile mode.

	Start Profiling (AMDTTeaPot.exe)	F5
	A <u>t</u> tach to Process	
~	Profile Mode - GPU: Performance Counters	
88	Pause Profiling	F6
	CPU: Assess Performance	
	CPU: Cache Line Utilization	
	CPU: Custom Profile	
	CPU: Instruction-based Sampling	
	CPU: Investigate Branching	
	CPU: Investigate Data Access	
	CPU: Investigate Instruction Access	
	CPU: Investigate L2 Cache Access	
	CPU: Time-based Sampling	
~	GPU: Performance Counters	
	Application Timeline Trace	
	Po <u>w</u> er Profiling	
	Select Power Profiling Counters	
	P <u>r</u> ofile Settings	

- 3. Click the (Start Profiling) toolbar button to start profiling.
- 4. Stop the profiled application when the part of the application that is under investigation has completed its

execution.

When the profiled application's execution is done, CodeXL displays the session.

	Method	ExecutionOrder	ThreadID	CallIndex	Glob	alWork	Size	Work	Gro	upSize	Time	LocalMemSize	VGPRs	SGPRs	ScratchRegs	FCStacks	KernelOccupan
1	horizontalSAT0 k1 Juniper1	1	8036	48	{ 1024	1024	1}	{ 256	1	1}	1.92811	0	15	NA	0	2	<u>66.6667</u>
2	horizontalSAT k2 Juniper1	2	8036	58	{ 1024	1024	1}	{ 256	1	1}	1.29789	0	15	NA	0	2	<u>66.6667</u>
}	horizontalSAT k2 Juniper1	3	8036	68	{ 1024	1024	1}	{ 256	1	1}	1.29533	0	15	NA	0	2	<u>66.6667</u>
ļ	horizontalSAT k2 Juniper1	4	8036	78	{ 1024	1024	1}	{ 256	1	1}	1.25700	0	15	NA	0	2	<u>66.6667</u>
;	horizontalSAT k2 Juniper1	5	8036	88	{ 1024	1024	1}	{ 256	1	1}	1.15156	0	15	NA	0	2	<u>66.6667</u>
;	verticalSAT k3 Juniper1	6	8036	98	{ 1024	1024	1}	{ 256	1	1}	1.17878	0	14	NA	0	2	<u>66.6667</u>
	verticalSAT k3 Juniper1	7	8036	108	{ 1024	1024	1}	{ 256	1	1}	1.23144	0	14	NA	0	2	<u>66.6667</u>
	verticalSAT k3 Juniper1	8	8036	118	{ 1024	1024	1}	{ 256	1	1}	1.28833	0	14	NA	0	2	<u>66.6667</u>
)	verticalSAT k3 Juniper1	9	8036	128	{ 1024	1024	1}	{ 256	1	1}	1.25600	0	14	NA	0	2	<u>66.6667</u>
.0	verticalSAT k3 Juniper1	10	8036	138	{ 1024	1024	1}	{ 256	1	1}	1.10333	0	14	NA	0	2	<u>66.6667</u>
1	box filter k4 Juniper1	11	8036	149	{ 1024	1024	1}	{ 256	1	1}	0.76067	0	6	NA	0	2	<u>100</u>
.2	box filter k4 Juniper1	12	8036	161	{ 1024	1024	1}	{ 256	1	1}	0.75745	0	6	NA	0	2	<u>100</u>
3	box filter k4 Juniper1	13	8036	173	{ 1024	1024	1}	{ 256	1	1}	0.75700	0	6	NA	0	2	<u>100</u>
.4	box filter k4 Juniper1	14	8036	185	{ 1024	1024	1}	{ 256	1	1}	0.75755	0	6	NA	0	2	<u>100</u>
5	box filter k4 Juniper1	15	8036	197	{ 1024	1024	1}	{ 256	1	1}	0.75744	0	6	NA	0	2	<u>100</u>
6	box filter k4 Juniper1	16	8036	209	{ 1024	1024	1}	{ 256	1	1}	0.75733	0	6	NA	0	2	<u>100</u>
17	box filter k4 Juniper1	17	8036	221	{ 1024	1024	1}	{ 256	1	1}	0.75733	0	6	NA	0	2	<u>100</u>

The GPU kernel performance counters can help find possible bottlenecks in the kernel execution. You can find the list of performance counters supported by AMD Radeon[™] GPUs in the tool documentation. Once we have used the trace data to discover which kernel most requires optimization, we can collect the GPU performance counters to drill down into the

kernel execution on a GPU device. Using the performance counters, you can:

- Find the number of resources (general-purpose registers, local memory size, and flow-control stack size) allocated for the kernel. These resources affect the possible number of in-flight wavefronts in the GPU. A higher number better hides data latency.
- Determine the number of ALU, global, and local memory instructions executed by the GPU.
- Determine the number of bytes fetched from, and written to, the global memory.
- Determine the use of the SIMD engines and memory units in the system.
- View the efficiency of the shader compiler in packing ALU instructions into the VLIW instructions used by AMD GPUs.
- View any local memory (Local Data Share LDS) bank conflicts. The Session View (see the screenshot above) shows the performance counters for a profile session. The output data is recorded in a comma-separated variable (.csv) format. You can also click on the kernel name entry in the "Method" column to view the OpenCL kernel source, AMD Intermediate Language (IL), GPU ISA, or CPU assembly code for that kernel.

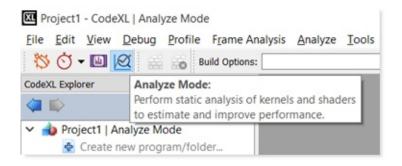
 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.



Switching to Analyze mode

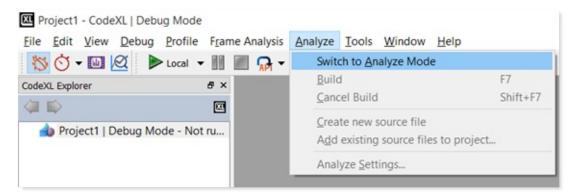
Option 1- Analyze mode button:

Click on the Analyze Mode button in the CodeXL Mode toolbar:



Option 2- Main menu:

Open the Analyze menu from menu bar and select the 'Switch to Analyze Mode' command:



After you switch to Analyze mode, you can also create a new project, open a previously saved project, or load the Teapot or

Matrix Multiply samples.

-

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.







Help > Tutorials > Static Analyzer Tutorial > Creating a new project for Analysis

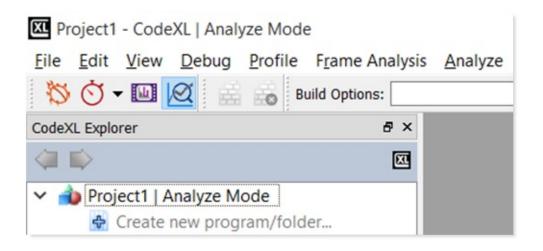
Creating a new project for Analysis

Click on the "File->Create Project", or use the Ctrl+N shortcut. The following CodeXL Project Settings dialog will appear:

Create a new CodeXL Project			
General	General		
V Profile	CodeXL Project Name:	Project2	
> CPU Profile	Target Host:	Local Host Remote Host	
Application Timeline Trace GPU Profile: Perf. Counters	Remote Host Address:		Port: 27015 Test Connection
Power Profile	Target Application		
Frame Analysis V Analyze	Application Type:	Desktop Application Windows Store App	
> Kernel/Shader Build Options	Executable Path:		i
HLSL Build Options	Working Directory:		
	Command Line Arguments:		
	Environment Variables:		
	Source Files		
	Source Files Directories:		
	Source Files Resolution Root Directory:		
Restore Default Settings			OK Cancel

Rename the project, and click on the OK.

After the new project has been created, the CodeXL Analyzer Explorer Tree should appear in the left pane:



 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.





Help > Tutorials > Static Analyzer Tutorial > Working with the new CodeXL Analyzer Explorer Tree

Working with the new CodeXL Analyzer Explorer Tree

If you are familiar with the former versions of the Analyzer, you probably noticed that the tree has a different structure than the one used in previous versions. Let's examine the structure of the new CodeXL Analyzer Explorer:

- 2. **Programs and Folders:** before describing how to technically create Programs and Folders, let's first discuss what those objects are, and why they can be useful.
 - c. Programs (OpenGL, Vulkan):

As of version 2.0, CodeXL can compile and link together multiple source files for OpenGL and Vulkan. This is especially important when different shaders have mutual impact on one another's ISA and performance statistics. To provide that type of support, CodeXL Analyzer introduced the concept of a Program. There are two types of Programs in CodeXL 2.0:

- Rendering Programs
- Compute Programs

A <u>Rendering Program</u> represents a graphics pipeline, and can have a single shader attached to each of its stages:

- Vertex
- Tessellation Control
- Tessellation Evaluation
- Geometry
- Fragment

A <u>Compute Program</u> represents a compute pipeline, and can have a single compute shader attached to its single stage.

When you build a program that has multiple shaders attached to it, all shaders are being compiled and linked together. This way, you get more accurate ISA and performance statistics than those generated using previous versions of CodeXL.

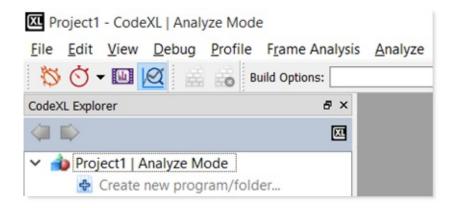
d. Folders (OpenCL, DirectX):

Folders are logical containers of source files. When you build a folder that has multiple source files attached to it, the source files are simply being built one after the other. Unlike programs, there is no kind of interdependency between the source files in a given folder: when a folder is being built, each source file is being compiled independently. Folders can be used to organize the project, by serving as a logical separator. They can also be used to ease the process of comparing build results, since now the build results are being maintained per-folder: you can create two different Folders, each containing the same source files, but have a different configuration (for example, create two DirectX Folders, each with a different shader model). After building the two Folders, you can toggle between the performance statistics of the two Folders to see the differences.

You may ask yourself why CodeXL does not support the concept of DirectX Programs, just like it does for OpenGL and Vulkan. This is a good point. Supporting DirectX Programs is at a high priority in the Analyzer's roadmap, and we will do our best to add that feature in the upcoming versions of the product.

Creating a new Program or Folder

To create a new Program or a Folder, double-click on the "Create new program/folder" item in CodeXL Analyzer Explorer Tree:

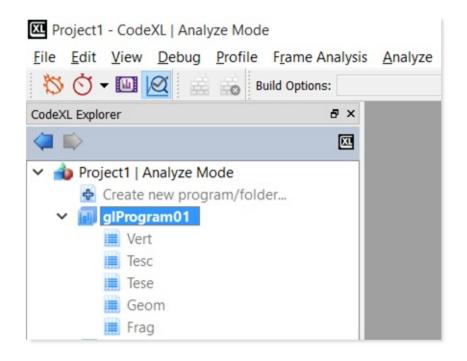


Then, the following dialog would pop-up:

Create a New Program/Folder	×
Select Program/Folder type:	
Vulkan	
Rendering Program	
Compute Program	
OpenGL	
Rendering Program	
O Compute Program	
DirectX	
O DirectX Folder	
OpenCL	
OpenCL Folder	
Description:	
An OpenGL Rendering Program is a graphics pipeline that can have a sin shader attached to each of its stages: Vertex, Tessellation Control, Tess Evaluation, Geometry, Fragment. When built, all of the program's shader being compiled and linked together.	ellation
ОК Са	ancel

Select the Program/Folder type of choice, and click OK.

Then, the empty Program/Folder would appear in the Explorer Tree. For Example, if you choose an OpenGL Rendering Program, you will see an empty OpenGL Rendering Program created:



 $\ensuremath{\textcircled{C}}$ 2015 Advanced Micro Devices. All rights reserved.

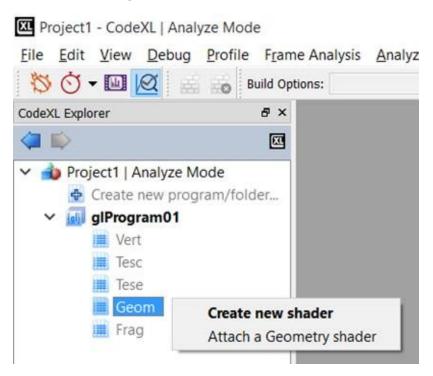




Help > Tutorials > Static Analyzer Tutorial > Working with Programs

Working with Programs

After creating a new program, you will see that it contains an empty placeholder for every pipeline stage. Right-click on any stage to add an existing shader or create a new one:



<u>Note</u>: You can also double-click on a stage to create a new shader and automatically attach it to that Program's stage.

As you can see in the above screenshot, we attached SimpleVertexShader.vs as the vertex shader to our OpenGL Rendering Program, and it was also automatically added to the Source Files pool. We can now drag SimpleVertexShader.vs from the Source Files pool and drop it on the stage node of any Program that we may add to the project, to reuse SimpleVertexShader.vs (there is no dependency in the build process between different Programs).

To build the program, right-click on it and select the Build option, or use the F7 shortcut:

Roject1 - CodeXL Anal	yze Mode	
File Edit View Debug	Profile Frame Analysis Analyze T	ools Window
🖏 🔆 • 🖬 🙋 🚊	Build Options:	
CodeXL Explorer	ð ×	
4		
 Project1 Analyze M Create new prog 		
✓ III glProgram01	Build glProgram01	F7
Vertex1.ve	Rename program	F2
Tese	Remove glProgram01 from Project	Del
Geometry	Export Binaries	
📕 Frag		
> 👩 Output (32-b	pit)	
> 📶 glProgram02		

You can also select the Program and manually click on the Build button in the Analyzer toolbar:

File Edit View Debug Profile Frame Analysis Analyze
🖏 🔆 🕶 🙋 🚔 🧓 Build Options:

© 2015 Advanced Micro Devices. All rights reserved.

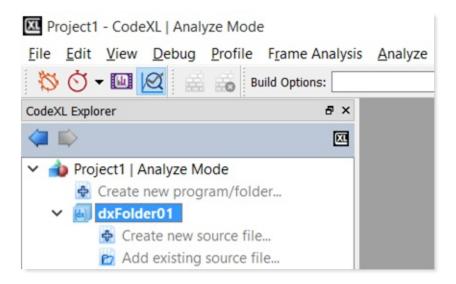




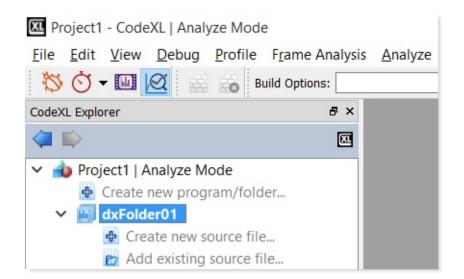
Help > Tutorials > Static Analyzer Tutorial > Working with Folders

Working with Folders

After creating a new OpenCL or DirectX Folder, an empty Folder would be listed in the Explorer Tree:



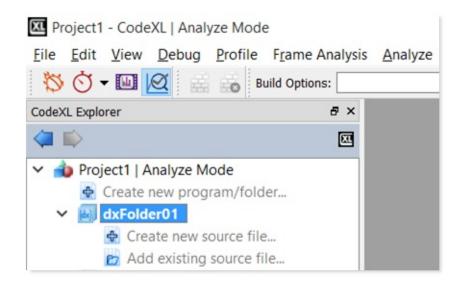
To create a new source file, and automatically add it to the Folder, double-click on the "Create new source file item..." item of the folder:



To add an existing source file, and automatically add it to the Folder, double-click on the "Add existing source file item..." item of the folder:







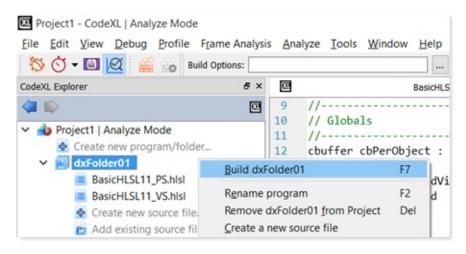
To configure the build properties of a source file under a specific Folder, click on that source file and use the Analyzer toolbar's Type and Entry point drop-down lists. The first sets the type of the shader and the latter specifies the specific target shader (among the shaders in the source file). This configuration is <u>Folder-specific</u>. That is, the same source file can be set with different properties under different Folders. CodeXL will remember those configurations for you.

To configure the build properties of the Folder, click on the Folder and adjust the enabled items in the Analyzer toolbar. For CodeXL 2.0, this is only relevant to the DX Shader Model property of DX Folders:

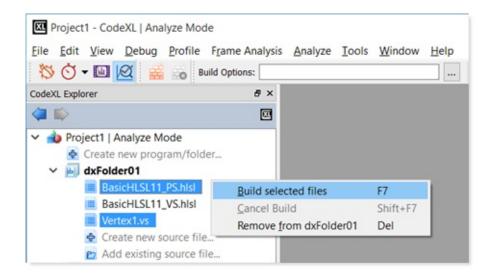


Once set, the DX Shader Model value will hold for all the shaders in the selected Folder. For example, if you choose 5_0 as the DX Shader Model, any D3D vertex shader in that Folder will be compiled using shader model vs 5_0 .

To build the whole Folder, right-click on it and select the Build item:



Unlike the case with Programs, Folders are more flexible as they allow you to build selected source files, without being required to build the whole Folder. To build selected source files, click on the selected source files under the program, while holding the Ctrl key. Then, right-click on one of the selected files and select the build option:



© 2015 Advanced Micro Devices. All rights reserved.

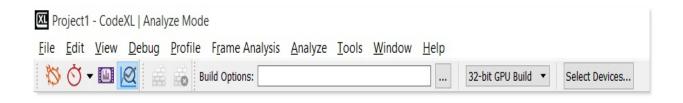




Help > Tutorials > Static Analyzer Tutorial > Selecting target devices

Selecting target devices

CodeXL Analyzer can target a variety of devices, independent of the device that is physically installed on your system. To select the target devices, for which the build would be performed, first click on the Select Devices button in the Analyzer toolbar:



Then, the CodeXL Options dialog would pop-pup with its Analyze tab activated. The devices are grouped by generations. You can use the check boxes to select and remove devices:

CodeXL Options	×
General GPU Debug CPU Profile Analyze	
Select devices to include them as build targets The displayed devices are devices supported by the installed driver	
Devices	
 > ✓ Carrizo > ☐ Fiji 	
> Iceland > Tonga	
Restore Default Settings OK	Cancel

© 2015 Advanced Micro Devices. All rights reserved.



Build Options- Defining OpenCL and DirectX build options

In the Static Analyze toolbar, you can define specific OpenCL or HLSL build options:

XL Pr	oject1	- Code	XL Anal	ze Mode	2						
<u>F</u> ile	<u>E</u> dit	<u>V</u> iew	<u>D</u> ebug	<u>P</u> rofile	Frame Analysis	<u>A</u> nalyze	<u>T</u> ools	Window	<u>H</u> elp		
Ø	Ò.		Q B	Bu	uild Options:					32-bit GPU Build 🔻	Select Devices

The Build Options box is a place to set compiler build flags such as $-x \operatorname{clc}^{++}$ or -03. Any compiler build flag can be placed in this box.

You can set the build options by typing the options directly in the designated text box or by using the OpenCL/HLSL Build Options dialog.

 $[\]ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.



Options- Defining OpenCL and DirectX build options

OpenCL and DirectX build

ar, you can define specific OpenCL or

<u>A</u> nalyze	<u>T</u> ools	<u>W</u> indow	<u>H</u> elp		
				32-bit GPU Build 🔻	Select Devices

place to set compiler build flags compiler build flag can be placed in

ns by typing the options directly in by using the OpenCL/HLSL Build

ed Micro Devices. All rights reserved.





Help > Tutorials > Static Analyzer Tutorial > The Statistics and Analysis Data

The Statistics and Analysis Data

Kernels Statistic view

The kernel statistics tab gives detailed statistics for the selected kernel for each target device. To open the statistics tab, expand the desired kernel in the project tree, and double-click the Statistics node:

Statistics page for devices GCN devices:

						- <u>-</u>					
Resource				Usage	-			(onstraint on N	10 fax Waves per SIM	D (1-10)
SGPRs (0-102) VGPRs (0-256)		24 Register 11 Register				10					
LDS size (0-32,768)	0/18 huter (Sta	atic: 2,048 bytes		uter)		8					
Amount of LDS available with	out reducing o			nic. 2,040 Dyte:	s, Dynamici V i	lytes)			0		
			Effec	tive concurre	ncy constraint	(Max waves	per SIMD):		8		
Resource	Reco	ommended Us	age		Usage			Performance In	npact		
ISA Size		<=32KB	-		608 bytes	;		Meets recommended usage			
Scratch Registers	h Registers 0				0			Meets recommended usage			
The effect of resource usage on the Max waves/SIMD: Num of SGPRs used:	10 <=48	9 49-56	8 57-64	7 65-72	6 73-80	5 81-96	4 96<				
Max waves/SIMD:	10	9	8	7	6	5	4	3	2	1	
Num of VGPRs used:	<=24	25-28	29-32	33-36	37-40	41-48	49-64	65-84	85-128	128<	
Max waves/SIMD:	10	9	8	7	6	5	4	3	2	1	
Amount of LDS used (bytes):	<=1,638	1,639-1,820	1,821-2,048	2,049-2,340	2,341-2,730	2,731-3,276	3,277-4,096	4,097-5,461	5,462-8,192	8,192<	
Applies to local workgroup dimension It is generally recommended to i to increased kernel performance The performance of kernels that	ncrease the nu e. t do a lot of mer	mber of waves i	n-flight. Hower	ver, this is not th enefit from havir		n-flight.		per of waves in-	flight does not a	lways translate	

This view is focused on giving the programmer the wave constraints based on the SGPRs, VGPRs and LDS size.

Shaders Statistic view

The shader statistics tab gives detailed statistics for the

selected shader for each target device.

To open the statistics tab, expand the desired shader in the project tree, and double-click the Statistics node:

Note: the statistics will be available only for V6 generation and later.

Shaders Statistics page:

esource						Usage	Constraint on Max Waves per SIMD (1-10)						
SGPRs (0-102)				4 Registers						10			
VGPRs (0-256)			3 Registers						10				
				Ff	Effective concurrency constraint (Max waves per SIMD):						10		
esource		Reco	mmende	d Usage			ι	lsage			Performance Impact		
ISA Size			<=32K	32KB 28 bytes							Meets recommended usage		
Scratch Registers			0					0			Meets recommended usage		
Performance Reference Ta	bles												
The effect of resource usa	ge on the n	umber of a	concurrent	waves									
Max waves/SIMD:	10	9	8	7	6	5	4						
Num of SGPRs used:	<=46	47-54	55-62	63-70	71-78	79-94	94<						
Max waves/SIMD:	10	9	8	7	6	5	4	3	2	1			
		25-28	29-32	33-36	37-40	41-48	49-64	65-84	85-128	128<			
Num of VGPRs used:	<=24	23-20											

This view is focused on giving the programmer the wave constraints based on the SGPRs and VGPRs.

© 2015 Advanced Micro Devices. All rights reserved.

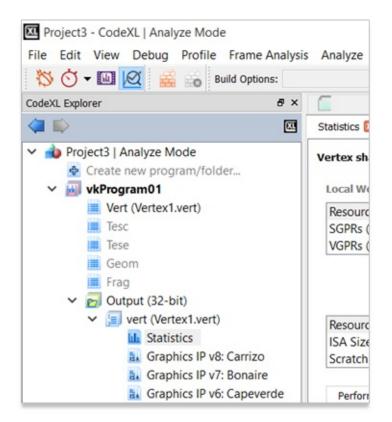




Help > Tutorials > Static Analyzer Tutorial > Viewing compilation output: IL and ISA

Viewing compilation output: IL and ISA

The performance statistics tab will be opened automatically when the build process is over. To view the compilation output, double click the node of the desired ASIC in the explorer tree, under the Program/Folder and configuration (32-bit or 64-bit):



This action will open a tab containing the source code, the AMD IL and the ISA:

Code								
	Source Code	IL Code -	calPriceVega	ISA Code - ca		00000000000		
286 🖒		1	<pre>mdef(16383)_out(1)_in(2)</pre>		V_RCP_F64	v[84:85] v[80:81]	16	Vector Arithmet
	calPriceVega(MonteCarloAttrib attrib,	2	mov r0, in0	0x001D10	V_CVT_F64_U32	v[80:81] v122	32	Vector Arithmet
88	int noOfSum,	3	mov r1, in1	0x001D24	V_RCP_F64	v[86:87] v[82:83]	16	Vector Arithmet
39	int width,	4	<pre>div_zeroop(infinity) r0.x, r0.x</pre>	0x001D70	V_CVT_F64_U32	v[82:83] v123	32	Vector Arithmet
90	global uint4 *randArray,	5	mov out0, r0	0x001D84	V_RCP_F64	v[88:89] v[84:85]	16	Vector Arithmet
91	global double4 *priceSam	6	mend	0x001DD0	V_MOV_B32	v76 0	4	Vector Arithmet
92	global double4 *pathDeriv)	7	il_cs_2_0	0x001DE4	S_AND_SAVEEXEC_B64	s[16:17] s[16:17]	4	Scalar Arithmet
93	{	8	<pre>dcl_cb cb0[15] ; Constant buffer t</pre>	0x001DF8	S_CBRANCH_EXECZ	label 07BA	4/16	Branch
94		9	dcl_literal 10, 0x00000004, 0x0000	0x001DFC	S_NOP	0x0000	1	Flow Control
95 96	<pre>double4 strikePrice = attrib.s</pre>	10	dcl_literal 11, 0x0000FFFF, 0xFFFF	0x001E00	V_MOV_B32	v82 0	4	Vector Arithmet
		11	call 1024;\$	0x001E14	V MOV B32	v82 0	4	Vector Arithmet
)7 10	<pre>double4 c1 = attrib.c1; double4 c2 = attrib.c1;</pre>	12	endmain		V CVT I32 F64	v76 v[76:77]	32	Vector Arithmet
8	<pre>double4 c2 = attrib.c2; double4 c3 = attrib.c3;</pre>	13	<pre>func 1024 ;OpenCL_calPriceVega_</pre>		V AND B32	v99 3 v76	4	Vector Arithmet
99 00	double4 c3 = attrib.c3; double4 initPrice = attrib.ini	14	dcl_max_thread_per_group 256	label_0784				
)))1	double4 sigma = attrib.sigma;	15	dcl_wavesize 64		S ANDN2 B64	exec s[16:17] exec	4	Scalar Arithmet
)2	double4 timeStep = attrib.time	16	dcl_indexed_temp_array x0[10]		S_CBRANCH_EXECZ	label 0857	4/16	Branch
)2	uouble4 climestep = acti 10.clime	17	mov x0[6], cb0[8].x		V_MOV_B32	v82 0	4/10	Vector Arithmet
)4	<pre>size_t xPos = get_global_id(0);</pre>	18 19	mov x0[0], 10.0000 mov x0[9], 10.0000		V_CNDMASK_B32	v82 0 v82 vcc	4	Vector Arithmet
)5	<pre>size_t yPos = get_global_id(1);</pre>	20	mov r0z_, vThreadGrpIdFlat0.x		V_FRACT_F64	v[84:85] v[84:85]	32	Vector Arithmet
06	are ,). as - 8c 8r 8r. (1),	20	mov r1022.xyz0, vTidInGrp0.xyz		V_CMP_GT_F64	vcc 0 v[92:93]	32	Vector Arithmet
)7	<pre>double4 temp = (double4)0.0;</pre>	22	mov r1023.xyz0, vThreadGrpId0.xyz			v93 0 v92 vcc	4	Vector Arithmet
08		23	dcl_literal 19, 256, 1, 1, 0xFFFFF		V_CNDMASK_B32			Vector Arithmet
09	double4 price1 = (double4)0.0;	24	umin r1023.xyz0, r1023.xyzz, 19.w		V_MOV_B32	v92 0	4	
10	<pre>double4 price2 = (double4)0.0;</pre>	25	umin r1021.xyz0, cb0[1].xyzz, l1.x		V_CVT_I32_F64	v92 v[92:93]	32	Vector Arithmet
11	<pre>double4 pathDeriv1 = (double4)0.0;</pre>	26	imad r1021.xyz0, r1023.xyzz, r1021		V_CVT_F64_I32	v[93:94] v92	32	Vector Arithmet
12	<pre>double4 pathDeriv2 = (double4)0.0;</pre>	27	umin r1024.xyz0, cb0[6].xyzz, 19.w		V_CVT_F64_I32	v[110:111] v99	32	Vector Arithmet
13		28	iadd r1021.xyz0, r1021.xyz0, r1024	0x002004	-	0x0000	1	Flow Control
14	<pre>double4 trajPrice1 = initPrice;</pre>	29	umin r1024.xyz0, cb0[7].xyzz, 19.w		V_AND_B32	v84 3 v84	4	Vector Arithmet
15	<pre>double4 trajPrice2 = initPrice;</pre>	30	iadd r1023.xyz0, r1023.xyz0, r1024	label_0857	' <mark>:</mark>			
16		31	mov r1023w, r0.z	0x00215C	S_MOV_B64	exec s[16:17]	4	Scalar Arithmet
17	<pre>double4 sumPrice1 = initPrice;</pre>	32	ishl r1023w, r1023.w, 10.z	0x002160	V_AND_B32	v85 1 v99	4	Vector Arithmet
18	double4 sumPrice2 = initPrice;	33	mov r1.x, 10.0000	0x002194	S_NOP	0x0000	1	Flow Control
19		34	mov x0[7], r1	0x0022F0	V_CMP_NE_I32	vcc 0 v86	4	Vector Arithmet
20	<pre>double4 meanPrice1 = temp;</pre>	35	udiv r1024.xyz_, r1021.xyzz, cb0[1	0x0022F4	V_CNDMASK_B32	v77 v82 v89 vcc	4	Vector Arithmet
1	<pre>double4 meanPrice2 = temp;</pre>	36	imad r1025.xyz0, r1023.xyzz, cb0[1	0x0022F8	V CNDMASK B32	v84 v83 v90 vcc	4	Vector Arithmet
2		37	mov x0[1], r1021		S XOR B64	vcc s[16:17] s[18:19]	4	Scalar Arithmet
3	<pre>double4 sumDeriv1 = temp;</pre>	38	mov x0[2], r1023		V_CNDMASK_B32	v86 0 v76 vcc	4	Vector Arithmet
24	<pre>double4 sumDeriv2 = temp;</pre>	39	dcl_cb_cb1[19]		V XOR B32	v84 v84 v86	4	Vector Arithmet
25		10	call 1026 . OnonCi calDeicellora			A		
26	<pre>double4 meanDeriv1 = temp;</pre>	(1	(III		•

Opening several build results of different devices for the same kernel/shader will open different IL/ISA tabs in the same "Source Code/IL/ISA" view as can be seen in the above image.

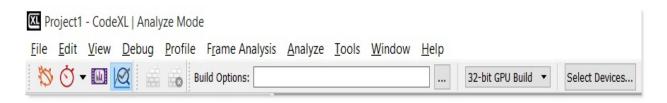
The context menu enables showing/hiding different parts of the view, enables showing/hiding line numbers, and enables different edit actions depending on the selection of where the context menu was opened and whether the view section is editable or not.

✓	Show OpenCL Sou	rce Code								
<	Show IL Code									
✓	Show ISA Code									
	Cut	Ctrl+X								
	Сору	Ctrl+C								
	Paste	Ctrl+V								
	Select All	Ctrl+A								
~	Show Line Numbe	rs								

© 2015 Advanced Micro Devices. All rights reserved.



In the Static Analyze toolbar, there is a space where you can define specific kernel/shader build options:



Build Options Dialog

This dialog will help you choose the correct kernel/shader build options for you and hopefully will prevent making spelling mistakes while typing the options manually.

To open the dialog, press The 🔜 Button. The dialog will be opened.

For OpenCL build options - you can browse between the "General & Optimization" tab and the "Other" tab to view all the available options.

For DirectX build options - choose the "HLSL Build Options" tab.

Once you choose an option, the option text is displayed in the text box marked ".. Build Command Line" that appears below. This string will also appear in the menu bar after you click the OK button.

Typing the option in the text box will also mark it in the appropriate area in the dialog, or select it in the appropriate

drop box.

CodeXL Project Settings			X
General	General		
Debug ✓ Profile	Predefined macros		
 CPU Profile Application Timeline Trace GPU Profile: Perf. Counters 	Additional include directories OpenCL format	Default	•
Power Profile Frame Analysis	Disable all warnings Treat any warning as an error	-w -Werror	
✓ Analyze	Optimization		
 Kernel/Shader Build Options General & Optimization 	Optimization level	Default	•
Other	Treat double float-point constant as single one	-cl-single-precision-constant	
HLSL Build Options	Flush denormalized floating point numbers as zeros	-cl-denorms-are-zero	
	Compiler assumes the strict aliasing rules	-cl-strict-aliasing	
	Enable MAD	-cl-mad-enable	
	Ignore the signedness of zero	-cl-no-signed-zeros	
	Allow unsafe optimization	-cl-unsafe-math-optimizations	
	Assume no NaN nor infinite	-cl-finite-math-only	
	Do aggressive Math Optimization	-cl-fast-relaxed-math	
	Correctly round single-precision FP divide sqrt	-cl-fp32-correctly-rounded-divide-sqrt	
	OpenCL Build Command Line		
Restore Default Settings			OK Cancel

DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of non-infringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. AMD, the AMD Arrow logo, AMD Radeon, AMD FirePro, gDEBugger, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Windows and DirectX are registered trademarks of Microsoft Corporation. PCIe and PCI Express are registered trademarks of PCI-SIG Corporation. OpenCL is a trademark of Apple Inc. Linux is a registered trademark of Linus Torvalds. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

© 2016 Advanced Micro Devices, Inc. All rights reserved.

[US] This should point to the glossary page title of the same name.

 $\ensuremath{\mathbb{C}}$ 2015 Advanced Micro Devices. All rights reserved.

[[]A1]Please review this cross-reference once and let me know whether it is correct.