

Nsight Eclipse Plugins Guide

Release 12.2

NVIDIA

Jul 25, 2023

Contents

1	About	Nsight Eclipse Plugins Edition	3
		Nsight Eclipse Edition Installing Nsight Eclipse Edition Installing CUDA Toolkit Configure CUDA Toolkit Path Nsight Eclipse Main Window Creating a New Project Importing CUDA Samples cuHook Sample 1 Configure Build Settings 1 Debugging CUDA Applications 1 Remote development of CUDA Applications 1 Improving Remote Debugging Performance 2 Profiling CUDA applications 2 Build CUDA Projects inside a Docker Container 2 Remote debugging using CUDA GDB inside Docker container	5 555788002230236
3	2.12 2.13	Importing Nsight Eclipse Projects 2 More Information 2 n Issues 3	
	 Notic 4.1 4.2 4.3 		33 33

Nsight Eclipse Plugins Edition Getting Started Guide

The user guide for using Nsight Eclipse Plugins Edition.

This guide introduces Nsight Eclipse Plugins Edition and provides instructions necessary to start using this tool. Nsight Eclipse is based on Eclipse CDT project. For a detailed description of Eclipse CDT features consult the integrated help "C/C++ Development User Guide" available from inside Nsight (through Help->Help Contents menu).

Chapter 1. About Nsight Eclipse Plugins Edition

NVIDIA[®] Nsight[™] Eclipse Edition is a unified CPU plus GPU integrated development environment (IDE) for developing CUDA[®] applications on Linux and Mac OS X for the x86, POWER and ARM platforms. It is designed to help developers on all stages of the software development process. Nsight Eclipse Plugins can be installed on vanilla Eclipse using the standard Help->Install New Software.. Menu. The principal features are as follows:

- Edit, build, debug and profile CUDA-C applications
- ▶ CUDA aware source code editor syntax highlighting, code completion and inline help
- ▶ Graphical user interface for debugging heterogeneous applications
- Profiler integration Launch visual profiler as an external application with the CUDA application built in this IDE to easily identify performance bottlenecks

For more information about Eclipse Platform, visit http://eclipse.org

Chapter 2. Using Nsight Eclipse Edition

2.1. Installing Nsight Eclipse Edition

Nsight Eclipse Plugins archive is part of the CUDA Toolkit. Nsight Eclipse Plugins archive can be installed using the Help -> Install New Software... Menu on Eclipse

2.1.1. Installing CUDA Toolkit

To install CUDA Toolkit:

- 1. Visit the NVIDIA CUDA Toolkit download page: https://developer.nvidia.com/cuda-downloads
- 2. Select appropriate operating system. Nsight Eclipse Edition is available in Mac OS X and Linux toolkit packages.
- 3. Download and install the CUDA Driver.
- 4. Download and install the CUDA Toolkit.
- 5. Follow instructions to configure CUDA Driver and Toolkit on your system.

2.1.2. Configure CUDA Toolkit Path

When Eclipse is first launched with Nsight Eclipse plugins in the new workspace, NVIDIA usage data collection dialog will be displayed as below. Click Yes to enable usage collection. This can be disabled later from the CUDA preference page.

To get started, CUDA Toolkit path must be configured in Eclipse with Nsight Plugins:

- 1. Open the Preferences page, Window > Preferences.
- 2. Go to CUDA toolkit section.
- Select the CUDA toolkit path to be used by Nsight. CUDA tookits that are installed in the default location will automatically appear.
- 4. CUDA toolkit path can be also specified in the project properties page in order to use different toolkit for a project.
- 5. Enable usage data collection if you wish to send usage data to NVIDIA.

Usage Data Collection



NVIDIA software packages collect usage and platform data via Google Analytics, including product version, display device, session data, and CPU information. If you agree, this data will be used anonymously for the sole purpose of improving NVIDIA software. For more information, please visit NVIDIA's Privacy Policy at https://www.nvidia.com/en-us/about-nvidia/ privacy-policy/.

Select "Yes" to enable usage collection.

Select "No" to disable usage collection.

No	Yes
----	-----

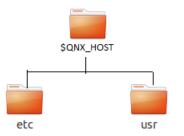
Fig. 1: Usage data collection page

😣 🗉 Preferences	_			
type filter text 🛛 🕿	CUDA			← < ⇒ < <
 General Ant C/C++ ChangeLog CUDA Docker Help Install/Update Java Library Hover 	CUDA Toolkit Target OS: Target Architecture: CCBIN CUDA Software Preen	/usr/local/cuda/bin Native x86-64 g++	• •	Add new SDK location Browse
Man pages CUDA software preemption enables single-GPU debugging and si Mylyn CUDA applications on the same device. Oomph Software preemption is supported on SM 3.5+ devices and not rest Plug-in Development Enable CUDA software preemption debugging Remote Development Enable CUDA software preemption debugging Remote Systems RPM				
 Run/Debug SystemTap Team Terminal Tracing Validation XML 	version, display devic anonymously for the	kages collect usage and platform data via Google An :e, session data, and CPU information. If you agree, t sole purpose of improving NVIDIA software. For mo cy at https://www.nvidia.com/en-us/about-nvidia/pr	his data v re inform	vill be used nation, please visit
	Make cuda-gdb and t	NVIDIA visual profiler as default launchers	Restore I	Defaults Apply
? 🖲 🕲			Cancel	Apply and Close

- 6. Click on the button to set cuda-gdb and Visual Profiler as the default launchers.
- 7. For QNX: When QNX is selected as Target OS, a dialog will be displayed to set the QNX_HOST and QNX_TARGET environment variables if they were not already set.

Set QNX_HOST and QNX_Target Environment Variables Set Environment Variable							
QNX_HOST:	/p4/sw/tools/embedded/qnx/qnx700-ga1/host/linux/x86_64	Browse					
QNX_TARGET: udeen/p4/sw/tools/embedded/qnx/qnx700-ga1/target/qnx7							
?	Cancel	ОК					

QNX_HOST environment variable identifies the directory that holds the host-related components:



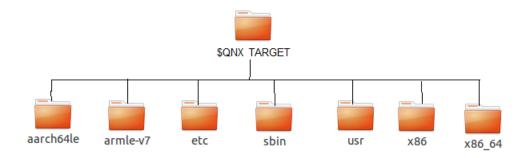
QNX_TARGET environment variable identifies the directory that holds the target-related components:

2.2. Nsight Eclipse Main Window

On the first run Eclipse will ask to pick a workspace location. The workspace is a folder where Nsight will store its settings, local files history and caches. An empty folder should be selected to avoid overwriting existing files.

The main Nsight window will open after the workspace location is selected. The main window is divided into the following areas:

▶ Editor - displays source files that are opened for editing.



- Project Explorer displays project files
- > Outline displays structure of the source file in the current editor.
- Problems displays errors and warnings detected by static code analysis in IDE or by a compiler during the build.
- ▶ Console displays make output during the build or output from the running application.

2.3. Creating a New Project

- 1. From the main menu, open the new project wizard File > New... > CUDA C/C++ Project
- 2. Specify the project name and project files location.
- 3. Specify the project type like executable project.
- 4. Specify the CUDA toolchain from the list of toolchains.
- 5. Specify the project configurations on the next wizard page.
- 6. Complete the wizard. The project will be shown in the Project Explorer view and source editor will be opened.
- 7. Build the project by clicking on the hammer button on the main toolbar.

2.4. Importing CUDA Samples

The CUDA samples are an optional component of the CUDA Toolkit installation. Nsight provides a mechanism to import these samples and work with them easily:

Note: Samples that use the CUDA driver API (suffixed with "Drv") are not supported by Nsight.

- 1. From the main menu, open the new project wizard File > New... > CUDA C/C++ Project
- 2. Specify the project name and project files location.
- 3. Select Import CUDA Sample under Executable in the Project type tree.
- 4. Select CUDA toolchain from the Toolchains option. location.

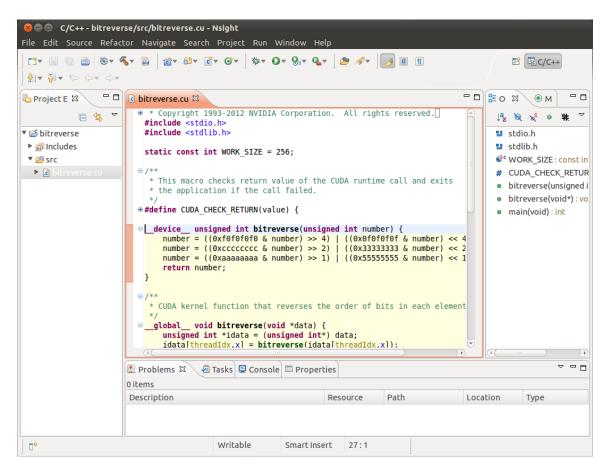


Fig. 2: Nsight main window after creating a new project

- 5. On the next wizard page select project sample you want to import. Also select the target CPU architecture. Press Next...
- 6. Specify the project parameters on the next wizard page.
- 7. Complete the wizard. The project will be shown in the Project Explorer view and source editor will be opened.
- 8. Build the project by clicking on the hammer button on the main toolbar.

2.4.1. cuHook Sample

cuHook sample builds both the library and the executable. cuHook sample should be imported as the "makefile" project using the following steps.

- 1. From the main menu, open the new project wizard File > New... > CUDA C/C++ Project
- 2. Select project type "Makefile project" and choose "Empty Project"
- 3. Specify the project name and project files location.
- 4. Complete the wizard. The project will be shown in the Project Explorer view.
- 5. Right click on the project Import... > General > File System
- 6. On the next wizard page, select the location of cuHook sample(Samples/7_CUDALibraries/cuHook)
- 7. Select all the source files and makefile and Finish the wizard
- 8. Build the project by clicking on the hammer button on the main toolbar.
- 9. To run the sample, from the main menu Run > Run Configurations... > Select the executable > Go to Environment tab > New... > enter Name=LD_PRELOAD, Value=./libcuhook.so.1 > Run will execute the sample

2.5. Configure Build Settings

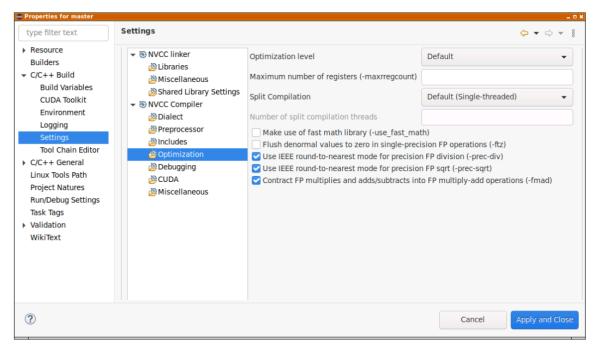
To define build settings: In the C/C++ Projects view, right-click your project, and select Properties. Select C/C++ Build, Settings from the list.

The following are the categories of Nvcc linker settings that can be configured for the selected project.

Note: All options field in the main page is not editable and it's the collection of options set in the child categories.

- Libraries Configure library search path(-L) and to include linker libraries(-l). When you are cross compiling for different target os, the library search path should point to the appropriate location where the target os libraries are present.
- ▶ Miscellaneous Set additional linker options and option to link with OpenGL libraries.
- Shared Library Settings Set option to build a shared library.

 Resource Builders C/C++ Build 	Configuration: Debug [Active]			
Build Variables CUDA Toolkit	< ऄ Tool Settings	ntainer Settings	🔎 Build Steps	Build Artifact
ECODA TOOKIL Environment Logging Settings Tool Chain Editor C/C++ General Linux Tools Path Project Natures Refactoring History Run/Debug Settings Task Repository Task Tags Validation WikiText	 NVCC linker Libraries Miscellaneous Shared Library Settin NVCC Compiler Dialect Preprocessor Includes Optimization Debugging CUDA Miscellaneous 	gs Command: All options: Expert settir Command line pattern:	arch=compute gencode arch=compute	c -ccbin g++ -gencode e_52,code=sm_52 - e_52,code=compute_52 D} \${FLAGS} \${OUTPUT_F
	L		Restore	e Defaults Apply
?			Cance	el Apply and Close



The following are the categories of Nvcc Compiler settings that can be configured for the selected project.

Note: All options field in the main page is not editable and it's the collection of options set in the child categories.

- > Dialect Select the language standard and dialect options.
- ▶ Preprocessor Add the defined and undefined symbols for the preprocessor.
- ▶ Includes Set include paths and include files for the compiler.
- Optimization Set the optimization level used by the compiler for code generation. Additionally, set the number of threads that the compiler will use during the compiliation process ("split compilation").
- > Debugging Set the options to generate debug information.
- CUDA Generate code for different real architectures with the PTX for the same vitrual architectures.

2.6. Debugging CUDA Applications

Nsight must be running and at least one project must exist.

- 1. In the Project Explorer view, select project you want to debug. Make sure the project executable is compiled and no error markers are shown on the project.
- 2. Right click on the project and go to Debug As > NVIDIA CUDA GDB Debugger menu.
- 3. You will be offered to switch perspective when you run debugger for the first time. Click "Yes". Perspective is a window layout preset specifically designed for a particular task.
- 4. Application will suspend in the main function. At this point there is no GPU code running.
- 5. Add a breakpoint in the device code. Resume the application.

Debugger will break when application reaches the breakpoint. You can now explore your CUDA device state, step through your GPU code or resume the application.

Additional debugger options can be set in the debug configuration dialog through Run > Debug Configurations .. menu..

2.7. Remote development of CUDA Applications

Nsight Eclipse Edition also supports remote development of CUDA application starting with CUDA Toolkit 6.0. The picture below shows how Nsight Eclipse Edition can be used for local as well as remote development:

For remote development you do not need any NVIDIA GPU on your host system. The remote target system can be a Linux x86 or POWER system with an NVIDIA GPU or an Tegra-based ARM system. Nsight IDE and UI tools can only be hosted on x86 and POWER systems.

Nsight Eclipse Plugins supports the cross compilation mode for remote devices.

‡ Debug ¤	💥 🕅 🚺 😵 🔻 🕯		🗱 🔨 😵 🕲 🗱	akpoints 🖗 Expressions 📃 🗖
 ▼ CUDA Thread (0,0) ■ BlackScholesGPI ▶ All Kernel Threads ▶ Host Process [black ↓ gdb] [device 0 (GK104GL)] (Breakpoint)	- 8	Name	Image: Second
<pre>//const int T const int opt // Calculati if (opt < (op { float cal float put BlackScho </pre>	<pre>tid = blockDim.x * blockIdx.x + threadIdx.x; ber of threads in execution grid HREAD_N = blockDim.x * gridDim.x; = blockDim.x * blockIdx.x + threadIdx.x; ng 2 options per thread to increase ILP (instruction</pre>	leve	0000000008fff58: 0000000008fff58: 0000000008fff58: 0000000008fff78: 0000000008fff78: 000000008fff79: 000000008fff99: 0000000008fff99: 0000000008fff88: 0000000008fff88: 0000000008fff88: 0000000008fff88:	Enterlocation here LDC R0, c[0x0][R0] MOV R3, c[0x0][0x20] SZR R4, SR CTAID.X MOV R4, R4 IMUL R3, R3, R4 SZR R4, SR TLD.X MOV R4, R4, R4 IADD R2, R3, R4 MOV R0, R0 MOV R3, R3 MOV R3, R3
🖳 Console 🛛 🧟 Tasks blackschole [C/C++ Applie	Problems Executables tation] blackschole			■ X ½ 5 11 8 6 6 1 0 • 11 • 1
Initializing data allocating CPU men generating GPU men generating input d copying input data Data init done.	ory for options. ata in CPU mem.			

Fig. 3: Debugging CUDA application

In the **cross compilation mode** the project resides on the host system and the cross compilation is also done on the host system. The cross compilation mode is only supported on an Ubuntu x86 host system.

To cross compile select the target cross compile architecture in CPU architecture drop down in the project properties page:

2.8. Debugging Remote CUDA Applications

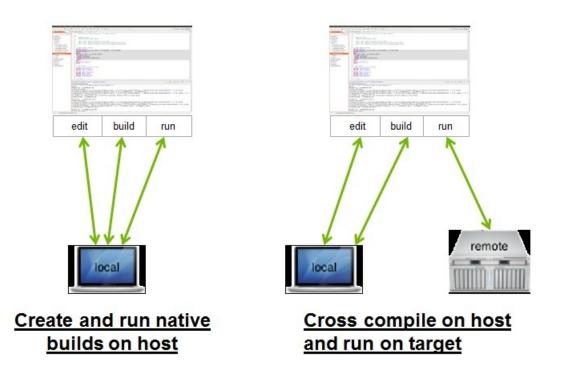
Remote debugging is available starting with CUDA Toolkit 5.5. A dedicated GPU is not required to use Nsight remote debugging UI. A dedicated GPU is still required on the debug target. Only Linux targets are supported. Debug host and target may run different operating systems or have different CPU architectures. The remote machine must be accessible via SSH and CUDA Toolkit must be installed on both machines.

Note: If there is a firewall between the host and the target, it must be set up to let RSP messages through, or SSH port-forwarding must be used.

 Select the project and right click then go to Debug As...>NVIDIA CUDA GDB Debugger(Remote) menu item.

😣 🗊 Debug Configurations					
Create, manage, and run configuration	IS		Ť		
 Image: Second Se	Name: blackschole Main ↔ Arguments Environme Stop on startup at: main Main Shared Libraries Break on CUDA API call failure Enable CUDA memcheck Print to console when CUDA ker Extrapolate "optimized out" CU Force thread list update on susp CUDA GDB executable: CUDA GDB init file: (Warning: Some commands in this f example "run".)	nel is launched or terminated IDA register values	Browse Browse e debugger, for		
Filter matched 17 of 27 items	Using CUDA GDB Create Process Laun	cher - Select other Revert	Apply		
0	111 - ATRAT - 11 - 11 - 7 - 11 - 7	Close	Debug		

Fig. 4: Debugging CUDA application



× + Properties for BlackScholes						
type filter text 🛛 🖾	CUDA Toolkit			← ▼ ⇒ ▼		
 ▶ Resource Builders ▼ C/C++ Build Build Variables CUDA Toolkit Environment Logging Settings Tool Chain Editor ▶ C/C++ General Run/Debug Settings ▶ Task Repository Task Tags ▶ Validation WikiText 	CUDA Toolkit Target OS: Target Architecture: CCBIN	/usr/local/cuda-9.0/bin Native ARM arm-linux-gnueabihf-g++	\$	Add new SDK location 32 Bit Browse		
?		Res	store De Cance			

Debug As	Þ	<u>1</u> Local C/C++ Application
Profile As	►	2 NVIDIA CUDA GDB Debugger
Compare With	•	3 NVIDIA CUDA GDB Debugger (Remote)
Restore from Local History		De <u>b</u> ug Configurations

2. Type the full path to a local executable or select one using the Local file... button.

😣 💷 Debug CUDA Application						
	Debug application on a remote system Run cuda-gdbserver remotely and connect debugger					
O Upload local exect	utable O Debug remote executable O D	o not transfer executable				
Local Executable	/blackschole/Debug/blackschole					
	(V	Norkspace				
Remote connection:	ubuntu@172.17.162.109	▼ Manage				
Remote executable:	/tmp/cuda-debug/blackschole	Browse				
Program arguments:						
cuda-gdbserver port:	2345 - +					
i ubuntu@172.17.162.109 is not connected. Remote paths will not be validated.						
?	< Back Next > C	ancel Finish				

- 3. Select a remote connection from a drop-down list or press the Add connection... button to create a new one.
- 4. If you are creating a new remote connection, enter the host name(or IP address) as well as the user name. Select the SSH as system type. Also select the QNX check box for QNX targets and then press Finish.
- 5. For Android devices: To configure the remote connection using Android debug bridge, select the Android debug bridge from the Remote Connection drop-down list, Android device must be connected to the host system using USB port.

Press Manage button, and enter or select the path to adb utility. You need to install Android SDK platform tools to use Android debug bridge. press Detect button to find the android device available through ADB.

- 6. **Optional:** Press Connect to verify the selected remote connection.
- 7. Press the Next button.
- 8. Type the full path to cuda-gdbserver on the remote system or select one using the Browse... button.
- 9. Click on "Add new path" or on the Browse... button to specify the path to the shared libraries the remote application depends on.

🗵 🔲 New Remote Connection

Remote Connections

Manage available connections

root@172.16	Add Remove
Host name:	172.16.174.115 QNX
User name:	root
Label:	root@172.16.174.115
System type:	SSH Port number: 22 - +
?	Cancel Finish

× + Debug CUDA Application							
Debug application on a remote system							
Run cuda-gdbserver remotely and connect debugger							
• Upload local exect	• Upload local executable O Debug remote executable O Do not transfer executable						
Local Executable	/async/Debug/async						
	W	orkspace Local File					
Remote connection:	Android debug bridge	▼ Manage					
Remote executable:	/data/local/tmp/async	Browse					
Program arguments:							
cuda-gdbserver port:	2345 – +						
i Android debug bridge is not connected. Remote paths will not be validated. Connect							
?	< Back Next > Car	rcel Finish					

×		
ADB Connec	tion	
Configure A	OB Connection	
ADB path:	deen/JetPack_Android2/android-sdk-linux/platform-tools/adb	Browse
ADB devices:	P2180A00P03600000505	Detect
_		
?	Cancel	ОК

Toolkit path:	/usr/local/cuda/	bin			Browse	Detect
library paths:						Browse.
	/usr/local/cud Add new path	a/lib				Delete
						Ŷ
						P
						-
i ubuntu@1	2.17.162.109 is no	t connected. Re	mote paths will	not be vali	idated.	Connect

- 10. Click on the Finish button to finish the new debug configuration wizard and start debugging the application.
- 11. You will be offered to switch perspective when you run the debugger for the first time. Click Yes. Perspective is a window layout preset specifically designed for a particular task.

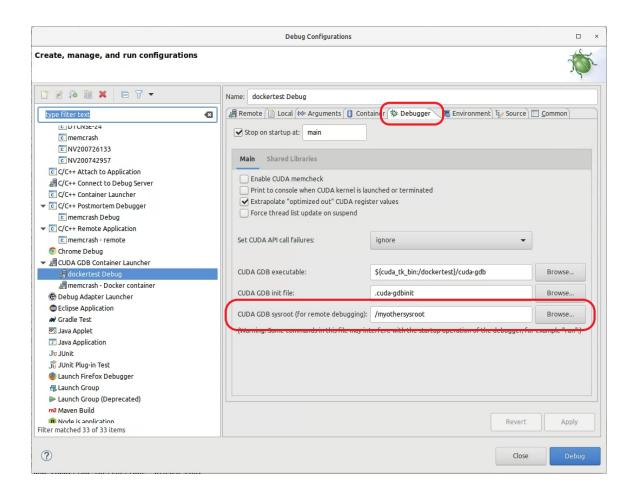
The debugger will stop at the application main routine. You can now set breakpoints, or resume the application.

😣 🖨 🗉 Debug - /home/test/bitreverse/bitreverse.cu - Nsight			
] ➡ ▼ 📓 💩] 券 ▼ O ▼ 9: ▼ 0 <u>↓</u> ▼] 🔌 া⊳ ।। 🔳 №] ⊿	3. O. A. 19	≅ ∞] 🕭 🖋 🕶	Ê ▼ ″
] st = 题 = th + + + + + + + + + + + + + + + + + +			
拳 Debug X 🕺 👔 👔 i i > マ 🗖 🗖	🕬= Variable 😫 🧕	ə Breakpo 🕅 CUDA 🚟 Re	gister 🛋 Modules 🛛 🗖 🗖
▼ ⓒ bitreverse on debugger-target [C/C++ Remote Application]		‱ ⇒t ; ⊟	8° × 🔆 🖬 🖻 🔻
 ▼ m Host Process [bitreverse] [cores: 0] ▼ m Thread [1] 10000 [core: 0] (Suspended : Breakpoint) 	Name	Туре	Value
■ miclo [1] 10000 [core: 0] (305)chice : Dreakpoint) ■ main() at bitreverse.cu:40 0x4027d0			
Remote Shell			
ji gdb			
🗈 bitreverse.cu 🕱		🗖 🗖 🔚 Outline	e X - D
<pre>* Host function that prepares data array and passes it to */</pre>	the CUDA kernel.	Ę	🗄 🞼 😿 🔏 🔹 🗰 🎽
<pre>int main(void) { bitreverse<<<1, WORK SIZE, WORK SIZE * sizeof(int)>>></pre>	().		
CUDA_CHECK_RETURN(cudaThreadSynchronize()); // Wait for		work to com	
CUDA_CHECK_RETURN(cudaGetLastError());	the dro fadhened		
CUDA_CHECK_RETURN(cudaDeviceReset());			
return 0; }			
📮 Console 🕱 🖉 Tasks 🕄 Problems 📀 Executables 🚺 Memory	= × 3		
bitreverse on debugger-target [C/C++ Remote Application] gdb			
Coalescing of the CUDA commands output is off.			
			1
] 0*			_ _

Fig. 5: Debugging remote CUDA application

2.8.1. Improving Remote Debugging Performance

When doing remote debugging, it can be useful to host copies of the target libraries in a local sysroot. Starting with CUDA 11.5, this feature is available on the **Debugger Configurations -> Debugger** tab. You can modify 'CUDA GDB sysroot (for remote debugging)' to point to a local sysroot directory to improve debugging performance.



2.9. Profiling CUDA applications

Nsight must be running and at least one project must exist. Profiler cannot be used when debugging session is in progress.

Nsight Eclipse Edition profiling features are based on the NVIDIA Visual Profiler (nvvp) code. Nsight Eclipse Plugins Edition will launch the Visual Profiler as an external tool with the executable and other information from the selected project.

- 1. In the Project Explorer view, select project you want to profile. Make sure the project executable is compiled and no error markers are shown on the project.
- 2. Select the project and right click and go to Profile As>NVIDIA Visual Profiler menu.

Profile As	•	<u>1</u> Local C/C++ Application
Compare With	•	<u>2</u> NVIDIA Visual Profiler
Restore from Local History		<u>3</u> NVIDIA Visual Profiler (Remote)
Run C/C++ Code Analysis –		Profile Configurations

Nsight Eclipse will launch the Visual Profiler to specify extra profiler options with the executable information already passed from the selected project.

File View Window Help				
	😣 🗉 Create New	Session		
	Executable Prope Set executable prop			
	Connection:	Local Cocal	nnections	
	Toolkit:	CUDA Toolkit 8.0 (/usr/local/cuda-8.0/bin/)	Manage	
	File:	/home/kjalaludeen/runtime-NsightPlugins3/blackschole/Debug/blackscho	Browse	
	Working directory:	[Enter working directory [optional]	Browse	
	Arguments:	Enter command-line arguments		
		Profile child processes	:	
	Environment:	Name Value	Add	
			Delete	
🛅 CPU Detail 🖾 📮 🗖				7 ° 0 • Pr 🛛 ° 0
1: 1: 5				
(Thread) 🔻		< Back Next > Cancel	Finish	Select or highlight a single interval to
Event	_		_	see properties

Fig. 6: Profiling CUDA Application

2.10. Build CUDA Projects inside a Docker Container

You can build and debug C/C++ and CUDA projects in a Docker container using Nsight Eclipse Edition. To get started, you need to first pull and install the Docker image that encapsulates the CUDA toolkit and cross platform tool chains. You can get the Docker images from NVIDIA GPU Cloud.Then you can use Nsight Eclipse Edition to build CUDA projects in a Docker container.

- 1. Open Nsight Eclipse Edition and configure the container settings.
- 2. Open the Preferences page, Window > Preferences and go to: CUDA > Container Settings

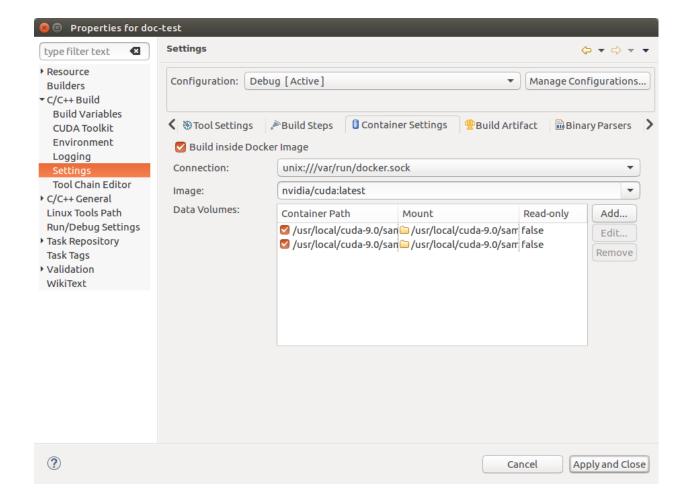
😣 🗉 Preferences		
type filter text	Container Settings	↔ → → →
 General Ant 	Build inside Docker Image	
 C/C++ ChangeLog CUDA 	Connection: unix:///var/run/docker.sock Image: hvidia/cuda:latest	•
Container Settings		
 Java Library Hover Man pages 		
 Mylyn Oomph Plug-in Development Remote Development 		
 Remote Systems RPM Run/Debug SystemTap Team Terminal 		
 Tracing Validation 		Restore Defaults Apply
? • ?		Cancel Apply and Close

- 3. Select the option if you want to build the projects inside the Docker container. Make sure the CUDA toolkit path that is specified in the CUDA preferences is the path of the CUDA toolkit inside a Docker container.
- 4. Select the Connection and the Image dropdown will display all the Docker images that are currently installed. Choose the docker image that you want to use to build/debug the projects. The preferences that are set here will be automatically displayed in the project setup wizard. You can choose to modify the container settings for the individual projects from the project setup wizard.
- To create a project, From the main menu, open the new project wizard: File > New... > CUDA C/C++ Project
- Specify the project name and project files location. And select the CUDA toolchain from the list of toolchains.

7. In the last page of project setup wizard, the container options will be displayed. The default container settings from the preference page will be displayed here. You can choose to modify the settings for this project in this Container settings page.

😣 🗊 C++ P	roject					
Container Settings Specify container settings for the project						
	ide Docker Ima					
Connection:	unix:///var/r	un/docker.sock			•	
Image:	nvidia/cuda:	latest			•	
?		< Back	Next >	Cancel	Finish	

- 8. Complete the project setup wizard. The project will be created and shown in the Project Explorer view.
- 9. The project source directories will be automatically mounted to the docker container.
- 10. If you need to mount any other directories that contains the include files/libraries and etc to the docker container, you can mount those directories from the project property page.
- Right click on the project and go to Properties. Select C/C++ Build > Settings > Container Settigns Tab. Additional directories can be mounted from this property page.
- 12. Build the project by hitting the hammer button on the main toolbar. The project is now built in the chosen Docker container the executable will be available on the host.



2.11. Remote debugging using CUDA GDB inside Docker container

From Nsight Eclipse, you can remote debug the applications running on the target using the CUDA GDB inside the Docker container running on the host. Docker images with CUDA GDB and CUDA toolkit must be already installed on the host. The remote machine must be accessible via SSH and CUDA Toolkit must be installed on target machine.

Note: If there is a firewall between the host and the target, it must be set up to let RSP messages through, or SSH port-forwarding must be used.

- 1. From the main menu, goto Run>Debug Configurations... menu item.
- 2. Create a new debug configuration under CUDA GDB Container Launcher either double clicking or using right click menu.
- 3. Configure the remote connection details in the "Remote" tab. If you are creating a new remote connection, click on the manage button in Remote Connection enter the host name(or IP address) as well as the user name. Select the SSH as system type. Also select the CUDA toolkit location on the target and choose the location to where to upload the executable.

🐱 Debug Configurations _ 💷 ×					
Create, manage, and run configurations		Ť			
	Name: Docker remote debug				
type filter text	📲 Remote 👔 Local 🖗 Arguments 🚺 Container 🕸 Debugger 🖾 Environment 🤴 Source 🔟 Common				
C/C++ Application	Remote connection: root@172.16.174.104	Manage			
C/C++ Attach to Application	Remote toolkit: /usr/local/cuda-10.0/bin	Manage			
C/C++ Container Launcher	Upload local executable Debug remote executable Do not transfer executable				
C C/C++ Postmortem Debugger C C/C++ Remote Application	Remote executable: /tmp/nsight-debug/app	Browse			
✓ I CUDA GDB Container Launcher	Debugger port: 2345 – +				
📕 Docker remote debug	Commands to execute remotely before starting debug server:				
Filter matched 17 of 18 items	Revert	Apply			
0	Close	Debug			

4. Choose the project and the executable to upload to the target in the "Local" tab.

5. From the "Container" tab, select the connection and Docker image that contains the CUDA GDB. Also you can select any host directories required to mounted to the Docker container.

e	Debug Configurations	- • ×
Create, manage, and run configurations		1. Alton
	Name: Docker remote debug	
type filter text 🛛 🛛	📲 Remote 📄 Local 🚧 Arguments 🚺 Container 🚿 Debugger 🖉 Environment 🦆 Source 🛅 Common	
C/C++ Application	Connection: unix:///var/run/docker.sock	-
C/C++ Attach to Application 場C/C++ Connect to Debug Server	Image: drive_os_qnx_sdk_new2:latest	-
C/C++ Container Launcher	Required host directories	New
C/C++ Postmortem Debugger	/home/kjalaludeen/Drive_5072/drive-t186ref-qnx/targetfs	
C/C++ Remote Application		Remove
 		
	Additional Options	<u> </u>
	Run in privileged mode	
Filter matched 17 of 18 items	Revert	Apply
?	Close	Debug

- 6. Select the debugger options from the "Debugger" Tab. Make sure to enter the CUDA GDB executable path inside Docker container. And add the required environment variables in the "Environment" tab.
- 7. Click on the Apply button to save the changes and click on the Debug button to start the debug session. This action will upload the local executable to the target system and will start CUDA GDB Server on the target. And the Docker container will be started on the host and CUDA GDB running inside the docker container will establish the remote debug session with the target.
- 8. You will be offered to switch perspective when you run the debugger for the first time. Click Yes. Perspective is a window layout preset specifically designed for a particular task.

The debugger will stop at the application main routine. You can now set breakpoints, or resume the application.

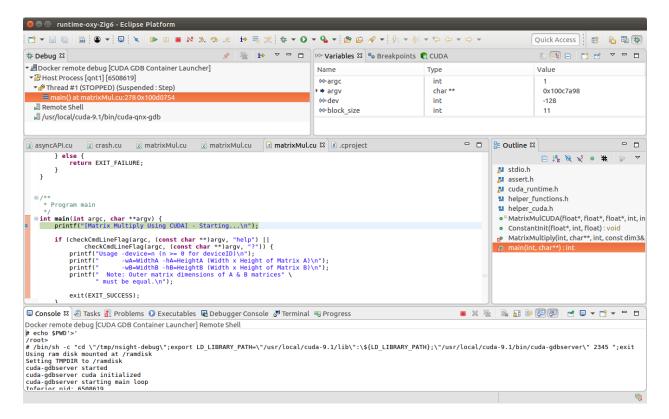


Fig. 7: Debugging remote CUDA application

2.12. Importing Nsight Eclipse Projects

The projects that are created with Nsight Eclipse Edition can be imported into the Eclipse workbench with Nsight Eclipse plugins.

- 1. Open Nsight Eclipse edition and select the project that needs to be exported.
- Right click on the Nsight Eclipse project and go to Export > C/C++ > C/C++ Project Settings > Next menu.
- 3. Select the project and settings to export.
- 4. Specify the "Export to file" location.
- 5. Settings will be stored in the given XML file.
- 6. Go to Eclipse workbench where the project settings needs to be imported.
- 7. Create a CUDA C/C++ Project from the main menu File > New > CUDA C/C++ Project
- 8. Specify the project name and choose Empty project type with CUDA toolchains.
- 9. Right click on the project to import the source files. Import > General > File System >(From directory) or copy the source files from the existing project.
- Import the project settings like include paths and symbols using the following right click menu Import > C/C++ > C/C++ Project Settings >Next...
- 11. Select the location of the project settigns file and select the project and configuration on the next wizard page.

× + Export				
Select				2
Select an export des	tination:			
type filter text				×
▶ 🗁 General ▼ 🗁 C/C++				
🕆 C/C++ Projec				
, Team Shared	Index			
🕨 🗁 Install				
?	< Back	Next >	Cancel	Finish

- 12. Complete the wizard. The project settings will be imported from the file exported from Nsight Eclipse Edition.
- 13. Build the project by clicking on the hammer button on the main toolbar.

2.13. More Information

More information about the Eclipse CDT features and other topics is available in the Help contents. To access Help contents select Help->Help Contents from the Nsight main menu.

More information about CUDA, CUDA Toolkit and other tools is available on CUDA web page at http: //developer.nvidia.com/cuda

× + Import	
Import	
Import C/C++ project settings	
Settings file /tmp/settings.xml	Browse
/ cmp/secongs.and	Drowsen
Select Project	Select Configuration
🗢 newp	🕥 🔉 Debug (Active)
😂 ssl	🛞 Release
😂 stl	
😂 twd	U
😂 wds	
Select settings to import	
🗹 🕒 Include Paths	0
Select All Deselect All	
? < Back	Next > Cancel Finish

Chapter 3. Known Issues

Executable must exist in order to start debug session for the first time

Nsight will not automatically perform build when starting debug session for a given project for the first time. Build must be invoked manually. Nsight will automatically rebuild executable when starting subsequent debug sessions.

Note: To manually build the project, select it (or any file within the project) in a Project Explorer view and click hammer icon on the main window toolbar.

Source editors may show error markers on a valid code for the files in newly created projects.

These markers will be cleared after Nsight indexes included header files.

Mac OS X users may be prompted to install Java Runtime Environment (JRE) when running Nsight Eclipse Edition for the first time.

Nsight Eclipse Plugin Edition requires functioning Java Runtime Environment to be present on the local system to run.

Nsight Eclipse Plugin Edition does not provide compilation support for using the QNX qcc and q++ compilers.

The workaround to compile using qcc and q++ is

- Specify the q++ path in CCBIN field on toolkit configuration page on project properties dialog as shown below. You can access toolkit configuration page by clicking main menu Project > Properties > C/C++ Build > CUDA Toolkit
- Change default CONF to gcc_ntoaarch64le in the file \${QNX_HOST}/etc/qcc/gcc/5.4.0/default as below

CONF=gcc_ntoaarch64le

Eclipse versions 4.26 through 4.28 are not presently supported.

Eclipse versions 4.26 through 4.28 are not presently supported due to a known compatibility issue with the Nsight Eclipse Edition plugins. The root cause of this compatibility issue has been identified, and a fix will be implemented in an upcoming release.

😣 🗊 Properties for tes	t			
type filter text 🛛 🕿	CUDA Toolkit			
 Resource Builders C/C++ Build Build Variables CUDA Toolkit Environment Logging Settings Tool Chain Editor C/C++ General Linux Tools Path Project Natures 	CUDA Toolkit Target OS: Target Architecture: CCBIN	\${QNX_HOST}/usr/bin/q++	Restore I	Add new SDK location Browse Defaults Apply
?			Cancel	Apply and Close

Chapter 4. Notices

4.1. Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or

services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARIS-ING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatso-ever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

4.2. OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

4.3. Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

©2016-2023, NVIDIA Corporation & affiliates. All rights reserved