

Intel® OpenCL SDK

User's Guide

Copyright © 2010-2011 Intel Corporation

All Rights Reserved

Document Number: 323626-001US



Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to <http://www.intel.com/design/literature.htm>

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to:
http://www.intel.com/products/processor_number

This document contains information on products in the design phase of development.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Intel, Intel logo, Intel Core, VTune, Xeon are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

OpenCL is trademarks of Apple Inc. used by permission by Khronos.

Copyright © 2010-2011 Intel Corporation. All rights reserved.



Contents

1	About Intel® OpenCL SDK	5
1.1	OpenCL* Overview	5
1.2	Intel® OpenCL SDK.....	5
2	Features	6
2.1	Full Coverage of the OpenCL* 1.1 Standard	6
2.2	Optional OpenCL* 1.1 Standard Features.....	6
2.3	OpenCL* 1.1 Optional Extensions	7
2.4	Intel® Vendor Extensions.....	7
2.4.1	Intel printf Extension Support	7
2.4.2	Intel Function Overloading Extension Support.....	8
2.5	OpenCL* Installable Driver Client (ICD) Compliant	8
2.6	Efficient, Highly Scalable Threading System	8
2.7	Smart Compilation Using the Implicit CPU Vectorization Module	9
2.8	Intel® VTune™ Amplifier XE 2011 Integration	9
2.9	Intel® OpenCL Offline Compiler.....	9
2.10	Intel® GPA Task Analyzer	9
3	Package Contents	10
4	Using the SDK	12
4.1	Using the OpenCL* runtime	12
4.1.1	Configuring Visual Studio* 2008.....	12
4.1.2	Using the OpenCL* Project's Property Pages.....	14
4.2	Building a Sample Application	15
4.3	Run The Sample Application.....	16
4.3.1	Using Visual Studio*	16
4.3.2	Using the Command Line.....	16
4.4	Working with the OpenCL* ICD	16
4.5	Working With the Intel® VTune™ Amplifier XE 2011	18
4.5.1	Setting a New Sampling Profiling Project.....	18
4.5.2	Viewing the OpenCL* Kernel's Assembly Code	19
4.6	Using the Implicit CPU Vectorization Module	20



4.7	Using the Intel® OpenCL SDK Offline Compiler	21
4.7.1	Building OpenCL* Kernels.....	21
4.7.2	Viewing the Generated Assembly Code	22
4.7.3	Viewing the Generated LLVM Code.....	23
4.7.4	Generating Intermediate Program Binaries.....	24
4.7.5	Saving the Code to a Text File.....	24
4.7.6	Working with the Command Prompt	24
4.8	Tracing OpenCL* Commands with the Intel® GPA Task Analyzer.....	25
4.8.1	About Intel® GPA Task Analyzer	25
4.8.2	OpenCL* Trace Layout	26
4.8.3	Using the Intel® OpenCL SDK Tracer	27
4.8.4	Generating a Trace File Manually	28
4.8.5	Task Analyzer Configuration File	28
4.9	Working with the <i>cl-fast-relaxed-math</i> Flag	30



1 *About Intel® OpenCL SDK*

1.1 OpenCL* Overview

OpenCL* is the first open, royalty-free standard for cross-platform, parallel programming of modern processors found in personal computers, servers and handheld/embedded devices. OpenCL* (Open Computing Language) greatly improves speed and responsiveness for a wide spectrum of applications in numerous market categories from gaming and entertainment to scientific and medical software.

1.2 Intel® OpenCL SDK

Intel® OpenCL SDK is an "Alpha" software implementation of the OpenCL* 1.1 standard optimized for Intel processors, running on Microsoft* Windows* 7 and Windows Vista* operating systems. The Alpha version of the SDK supports the **CPU device only** and is intended for both 32-bit and 64-bit applications. For a full list of supported platforms, please refer to the Intel® OpenCL SDK Release Notes document.

The Intel® OpenCL SDK delivers a complete solution that enables users to create applications that include OpenCL* 1.1 API calls and kernels.

The Intel® OpenCL SDK is available on whatif.intel.com. On the "whatif" web site, Intel provides "Alpha" software free of charge, but without any warranty that it works as expected or will be supported in its current state in the future. Since it is only a technology preview, the product may be incomplete and not tested sufficiently for productive usage. Nevertheless, this release gives you the opportunity to experiment with new capabilities long before they become available as a commercial product



2 Features

2.1 Full Coverage of the OpenCL* 1.1 Standard

The Intel® OpenCL SDK supports the complete OpenCL* 1.1 language and an Application Programming Interface (API). This version of the SDK was validated with Khronos OpenCL* conformance tests suite and passed all of them. For more information on this version conformance status please refer to Intel® OpenCL SDK website.

For information on the product's limitations and known issues please refer to the **Intel(R) OpenCL* Release Notes**.

2.2 Optional OpenCL* 1.1 Standard Features

The Intel® OpenCL SDK supports the following OpenCL* 1.1 optional features:

- Out-of-order Execution model as defined in the OpenCL specification (CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE property of a command-queue).
- Execution of native kernels, as defined in the specification (CL_EXEC_NATIVE_KERNEL option of the CL_DEVICE_EXECUTION_CAPABILITIES property of a device info).
- Image support with the minimum set of image formats, as defined in the specification (CL_DEVICE_IMAGE_SUPPORT property of a device info).
- Optimization Options: The SDK supports the OpenCL* 1.1 standard compiler flag **-cl-fast-relaxed-math**. Some optimizations may violate the IEEE 754 standard and the OpenCL* numerical compliance. For a full list of optimized functions see [Working with the cl-fast-relaxed-math Optimization Flag](#)
- Math intrinsic option: The SDK supports the OpenCL* 1.1 standard optional compiler flag **-cl-denorms-are-zero**.



2.3 OpenCL* 1.1 Optional Extensions

The Intel® OpenCL SDK supports the following OpenCL* 1.1 extensions:

- `cl_khr_fp64` - Double precision floating point support, as defined in the specification
- `cl_khr_gl_sharing` - Creating CL context from a GL context or share group, as defined in the specification
- `cl_khr_gl_sharing` - Sharing memory objects with OpenGL or OpenGL ES buffers, texture and render bugger objects, as defined in the specification

2.4 Intel® Vendor Extensions

2.4.1 Intel printf Extension Support

The Intel® OpenCL SDK supports the `cl_intel_printf` extension, which enables the kernel developer to use the C `printf` function with support for OpenCL vector types. The `printf` function can be seen as a new OpenCL builtin with the following signature:

```
int printf(__constant char* restrict format, ...);
```

The semantics of `printf` match the definitions found in section 7.19.6 of the C99 standard, with the following notes:

- 64-bit integer types can be printed using the `l` (lowercase L) length modifier (e.g. `%lu`)
- The `ll` length modifier isn't supported

OpenCL vector types can be explicitly passed and printed using the modifier `vn` where `n` can be 2, 3, 4, 8 or 16. This modifier appears before the original conversion specifier. I.e. to print a `int4` the conversion specifier is `%v4d`. A comma is used as the separator between the printed vector components.



2.4.2 Intel Function Overloading Extension Support

The Intel® OpenCL SDK supports the **cl_intel_overloading** extension, which enables the kernel developer to use overloadable functions. User overloadable functions are a multiple standard C functions that can share the same name. Functions of the same name must differ by operand type and/or count. Functions are declared overloadable if they use the overloadable attribute. The overloadable attribute may appear before the function name or after the closure of the operand list.

Examples:

```
float my func( float ) attribute ((overloadable));
```

```
double __attribute__ ((overloadable)) my func( double );
```

```
int my func( float16, constant char* ) attribute ((overloadable));
```

2.5 OpenCL* Installable Driver Client (ICD) Compliant

The OpenCL* Installable Client Driver (ICD) enables multiple OpenCL* implementations to coexist under the same system. It also enables applications to select between them at runtime.

The Intel® OpenCL SDK supports the official OpenCL* 1.1 ICD. See [Working with OpenCL* ICD](#) for more details.

2.6 Efficient, Highly Scalable Threading System

The Intel® OpenCL SDK contains an efficient, highly scalable threading system for optimal multi-core performance. The system is based on Intel® Threading Building Blocks (Intel® TBB) technology.



2.7 Smart Compilation Using the Implicit CPU Vectorization Module

The Intel® OpenCL SDK provides a smart OpenCL* compilation environment that efficiently maps the code into the CPU's vector units and includes a unique implicit CPU vectorization module for best utilization of the hardware vector units across work items. The vectorization module aims to merge together the execution of several work items, utilizing the Intel vector instruction set.

The compilation environment is based on the open source LLVM compiler and its clang front end.

2.8 Intel® VTune™ Amplifier XE 2011 Integration

The SDK provides the ability to analyze the assembly code of the OpenCL* kernel with the Intel® VTune™ Amplifier XE 2011. For more information, see [Working With the Intel® VTune™ Amplifier XE 2011](#).

2.9 Intel® OpenCL Offline Compiler

The Intel® OpenCL SDK provides a unique standalone tool that offers full offline OpenCL* language compilation including the OpenCL* syntax checker, LLVM viewer, Assembly language viewer and intermediate program binaries generator. See [Using the Intel® OpenCL Offline Compiler](#) for more details.

2.10 Intel® GPA Task Analyzer

The Intel® OpenCL SDK provides a standalone tracing tool that enables developers to see the OpenCL* kernel's' execution data in a trace. You can analyze the trace offline to enhance the performance of your OpenCL* application. See Tracing [OpenCL* Commands with the Intel® GPA Task Analyzer](#) for more details. The Intel® OpenCL SDK also provides the **OpenCL SDK Tracer**, a tool that enables you to easily launch an OpenCL* application to automatically generate trace files. See [Using the Intel® OpenCL SDK Tracer](#) for more details.



3 Package Contents

The Intel® OpenCL SDK installation package contains the SDK libraries, documentation, tools, developer files and samples. During the installation you can choose the installation path for the libraries and the samples.

The default installation path is: `C:\Program files\Intel\OpenCL SDK` on 32-bit systems and `C:\Program files (x86)\Intel\OpenCL SDK` on 64-bit systems.

The default installation path for the samples package is: `C:\Users\<user name>\Documents\Intel\OpenCL SDK`

Where **<user name>** is the name of the local user account that is installing the SDK.

Each folder within the installation directory contains the following directories:

- `bin`
Contains the OpenCL*-related binaries
- `lib`
Contains the basic OpenCL* runtime library and the OpenCL* ICD library, to which the applications should link.
- `include`
Contains relevant header files, including the headers for the OpenCL* runtime
- `doc`
Includes related documents
 - *Intel(R) OpenCL SDK Installation Notes.pdf*
Intel® OpenCL SDK installation notes document.
 - *Intel(R) OpenCL SDK User Guide.pdf*
Intel® OpenCL SDK overview and usage documentation.
 - *Intel(R) OpenCL SDK Release Notes.pdf*
Intel® OpenCL SDK release notes document.
 - *Writing Optimal OpenCL* Code with Intel(R) OpenCL SDK.pdf*
Intel® OpenCL SDK optimization guide document.



- **tools**

Includes the tools that come with the SDK. The tool-set includes:

- the Intel® OpenCL Offline Compiler
- the Intel® Task Analyzer SDK

- **samples**

Includes sample projects and property pages. The samples package includes the following samples:

- *Dot Product*
Demonstrates how to compute a dot product of two float4 arrays and writes the result to a float array
- *Bitonic Sort*
Demonstrates how to sort an arbitrary input array of integer values with OpenCL* using Single Instruction Multiple Data (SIMD) bitonic sorting networks.
- *God Rays*
Demonstrates how to use high dynamic range (HDR) rendering with God Rays (crepuscular rays) in OpenCL*.
- *Median Filter*
Demonstrates how to use a median filter in OpenCL*.



4 Using the SDK

4.1 Using the OpenCL* runtime

The Intel® OpenCL SDK related binaries are installed to the following directory:

`$(INTELOCLSDKROOT)\bin\x86`

`$(INTELOCLSDKROOT)\bin\x64` On 64-bit operation systems only

Where **INTELOCLSDKROOT** is an environment variable that is automatically added to the system during installation and points to the directory which the SDK was installed to.

This directory is automatically added to the system PATH environment. See the Installation Notes for more information.

In order to be able to work with the OpenCL* runtime, an application should link to the OpenCL* import library, `intelocl.lib`, which contains an import address table (IAT). The OpenCL* runtime DLL function calls are referenced by this table, or through the OpenCL* Installable Client Driver (ICD). Both libraries are installed to:

`$(INTELOCLSDKROOT)\lib\x86`

`$(INTELOCLSDKROOT)\lib\x64` On 64-bit operation systems only

For more information on how to use the OpenCL* ICD, see [Working with OpenCL* ICD](#).

4.1.1 Configuring Visual Studio* 2008

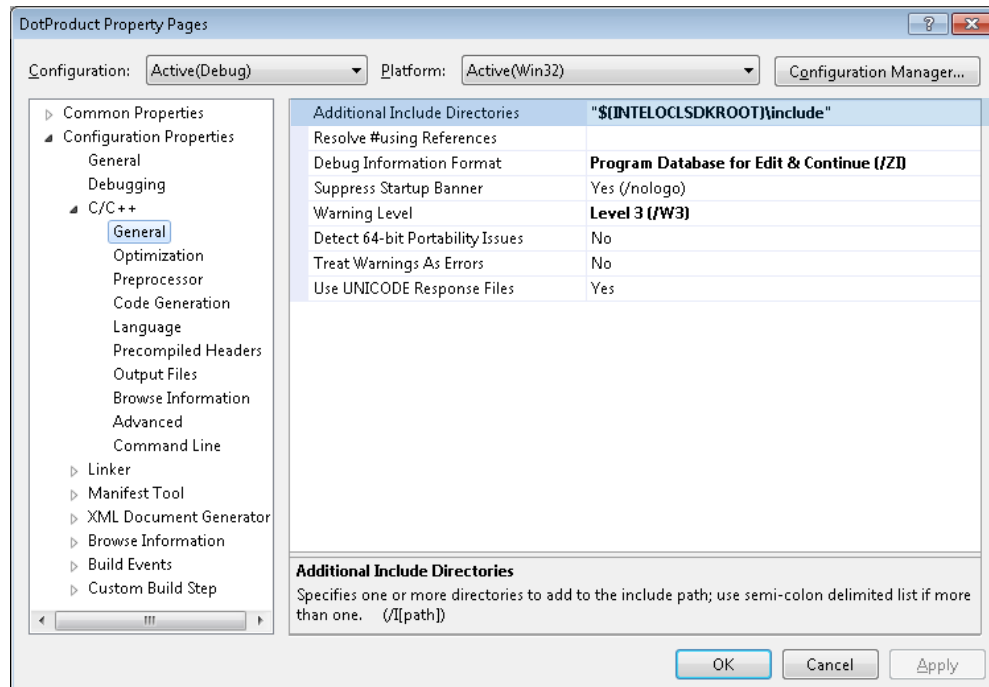
To setup a Visual Studio* 2008 project to work with the SDK:

1. Open the project's property pages by selecting **Project > Properties**,

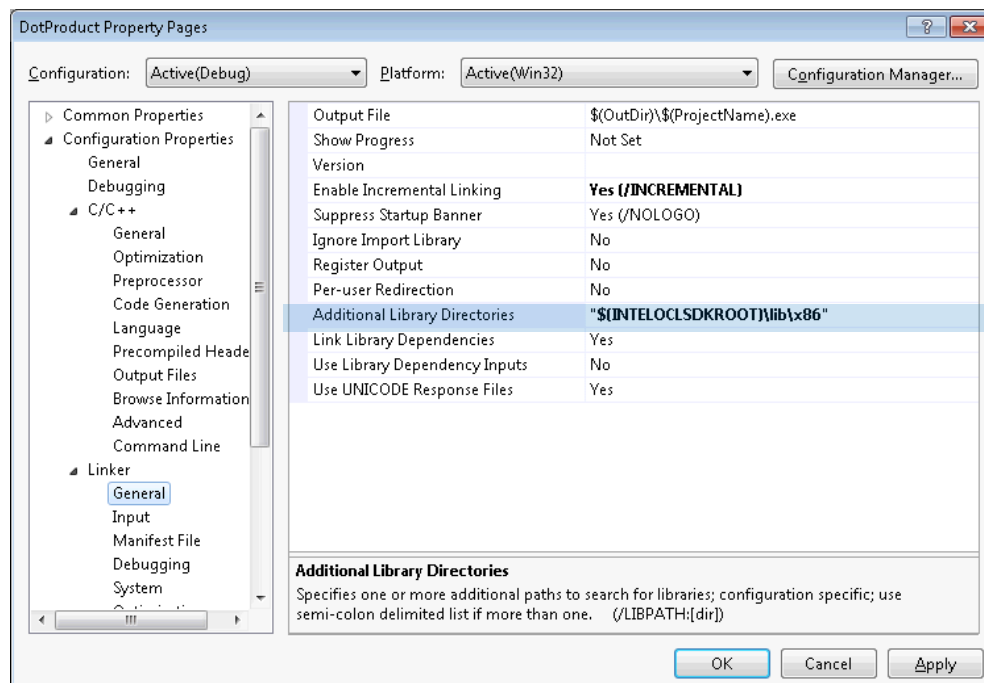
In the **C/C++ > General** property page, under **Additional Include Directories**, enter the full path to the directory where the OpenCL* header files are located:



`$(INTELOCLSDKROOT)\include.`

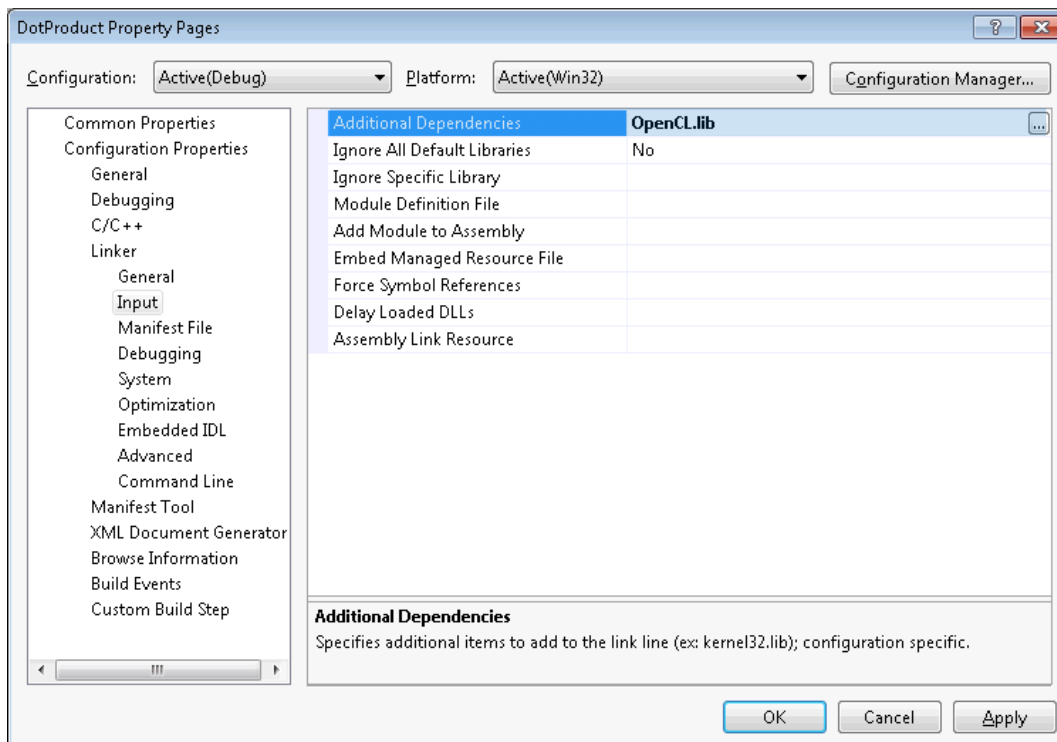


2. In the **Linker > General** property page, under **Additional Library Directories**, enter the full path to the directory where the OpenCL* runtime import library file is located. For example, for 32-bit application: **`$(INTELOCLSDKROOT)\lib\x86.`**





3. In the **Linker > Input** property page, under **Additional Dependencies**, enter the name of the OpenCL* runtime import library file **intelocl.lib**. If you want to use the OpenCL* ICD, enter the ICD library name: **OpenCL*.lib** instead.



4.1.2 Using the OpenCL* Project's Property Pages

A project property page is a Visual Studio* user interface element that enables you to specify and apply project settings to your projects.

The Intel® OpenCL SDK provides such templates with the required settings so you can easily create OpenCL* applications. The templates were generated for Microsoft® Visual Studio* 2008.

In order to specify your project properties with the property page:

1. Create a new Visual Studio* project
2. Open your project's Property Pages dialog box.
3. Using the **Configuration** dialog box, select the configuration for which you want to set this project's properties, for example:
intelocl_win32_debug.vsprops for 32-bit debug configuration and
intelocl_win32_release.vsprops for 32-bit for release configuration)



4. Click OK to save the settings. If you want a change to take effect immediately, click Apply.

The property page files are installed to the following directory:

`$(INTELOCLSAMPLESROOT)\templates`

4.2 Building a Sample Application

The Intel® OpenCL SDK installation package contains a Microsoft* Visual Studio* 2008 solution file, `OpenCL Samples.sln`. The file is located at

`$(INTELOCLSAMPLESROOT)\`

Where **INTELOCLSAMPLESROOT** is an environment variable that is automatically added to the system during the installation and points to the SDK samples package installation directory.

The solution file contains the following Microsoft* Visual Studio* 2008 sample projects: `DotProduct`, `BitonicSort`, `GodRays`, `MedianFilter` and `ShallowWater`.

In order to build all samples:

1. Open `OpenCL Samples.sln`.
2. Select **Build > Build Solution**.

In order to build a specific project:

1. Open `OpenCL Samples.sln`.
2. Right click on the project file in the Solution Explorer's tree view.
3. Select **Build**.



4.3 Run The Sample Application

There are two ways to run the sample application: using Microsoft Visual Studio* 2008 Professional Edition, or using the command line.

4.3.1 Using Visual Studio*

1. Open and build `OpenCL Samples.sln`.
2. Select a project file in the **Solution Explorer**.
3. Right-click the project and select **Set as StartUp Project**.
4. Press Ctrl+F5 to run the application. To run the application in debug mode, press F5.

4.3.2 Using the Command Line

1. Open a Visual Studio* command prompt.
2. Change to the directory according to the platform configuration on which you want to run the executable:

`$(INTELOCLSAMPLESROOT)\Win32` for Win32 configuration

`$(INTELOCLSAMPLESROOT)\x64` for x64 configuration

3. Open the appropriate project configuration (**Debug** or **Release**).
4. Run the sample by entering the name of the executable. For example run `DotProduct.exe` in order to run the Dot Product sample.

4.4 Working with the OpenCL* ICD

When using the OpenCL* ICD, the application is responsible for selecting the OpenCL* platform to use. If there are several OpenCL* platforms installed on the system, the application should use the `clGetPlatformIDs` and `clGetPlatformInfo` functions to query the available OpenCL* platforms and decide which one to use.

The following code snippet shows an example of how to query the system platforms to get the correct platform ID:



```
cl_platform_id * platforms = NULL;
char vendor_name[128] = {0};
cl_uint num_platforms = 0;

// get number of available platforms
cl_int err = clGetPlatformIDs(0, NULL, & num_platforms);
if (CL_SUCCESS != err)
{
    // handle error
}
platforms = (cl_platform_id*)malloc(
    sizeof(cl_platform_id) * num_platforms);
if (NULL == platforms)
{
    // handle error
}
err = clGetPlatformIDs(num_platforms, platforms, NULL);
if (CL_SUCCESS != err)
{
    // handle error
}

for (cl_uint ui=0; ui< num_platforms; ++ui)
{
    err = clGetPlatformInfo(platforms[ui],
        CL_PLATFORM_VENDOR,
        128 * sizeof(char),
        vendor_name,
        NULL);
    if (CL_SUCCESS != err)
    {
        // handle error
    }
    if (vendor_name != NULL)
    {
        if (!strcmp(vendor_name, "Intel Corporation"))
        {
            return platforms[ui];
        }
    }
}

// handle error
```



4.5 Working With the Intel® VTune™ Amplifier XE 2011

The Intel® OpenCL SDK can work with the Intel® VTune™ Amplifier XE 2011 to locate performance bottlenecks and analyze the performance results of your OpenCL* application.

The SDK enables you to see the assembly code of your OpenCL* kernels JIT code and to analyze its performance through sampling profiling (call-graph profiling is not supported in this version) using the graphical interface of the Intel® VTune™ Amplifier XE performance profiler.

The Intel VTune™ Amplifier XE 2011 is a separate utility that need to be acquired separately. For more information, please visit the [product web site](#)

NOTE: The profiling support of Intel® OpenCL SDK is designed to work with the Intel® VTune™ Performance Analyzer 9.1 (Build: 385) or later. Stable work with a different version is not guaranteed.

4.5.1 Setting a New Sampling Profiling Project

NOTE: The instruction below and all screen shots refer to the Intel® VTune™ Performance Analyzer 9.1 (Build: 385)

To run sampling profiling on the VTune™ analyzer:

1. Create a new Sampling Project by selecting **File > New Project... > Sampling Wizard**
2. Select the **Windows*** type of profiling in the “**Select type of profiling**” group box click **Next**.
3. Select the application to run, including the working directory and command argument. Make sure that the **Run Activity when done with wizard** check-box is **NOT** checked. Click **Next**.
4. Click **Finish** to complete the wizard.
5. Select the new project in the Tuning browser and show its properties by selecting **Activity > Modify**. Select **Modify the selected Activity**.
6. In the **Advanced Activity Configuration** form select the name of your application and click **Configure....**



7. In the **Application/Module Profile Configuration** form click **Advanced**.
8. Uncheck the **Use default environment** check-box and add these 2 lines to the **Environment Variables** text-box:
ENABLE_JITPROFILING=1
CL_CONFIG_USE_VTUNE=True
9. Click **OK** in all opened forms to save new settings
10. Click the **Run Activity** button to run and analyze your application.

4.5.2 Viewing the OpenCL* Kernel's Assembly Code

When the sampling activity finishes, open the activity's sampling results through the tuning browser and look for the application's process in the **Processes** data grid:

		sample	samples	per...	%	%	events	events
MyApp.exe		282,162	198,184	1.424	99.75%	99.81%	865,108,692,000	607,632,144,000
cmd.exe		148	83	1.783	0.05%	0.04%	453,768,000	254,478,000

Select the application and click the **Display Module for Selected Items** button to show the process' modules.

Look for the **Application_Name.jit** module in the **Modules** data grid:

MyApp.exe.jit	MyApp.exe	280,273	196,856	1.424	99.33%	99.33%	859,317,018,000	603,560,496,000
cmd.exe	cmd.exe	148	83	1.783	0.05%	0.04%	453,768,000	254,478,000

Select the process's JIT code module on the **Display Hotspots for Selected Items** button to show the module's OpenCL* kernels:

Name	CPU_sample	INST_R samples	Clocks per...	CPU_CL %	INST_R %	CPU_CLK_UNHA events	INST_RETIRED. events
ocl_kernel2	133,480	93,715	1.424	47.62%	47.61%	409,249,680,000	287,330,190,000
ocl_kernel4	133,288	93,681	1.423	47.56%	47.59%	408,661,008,000	287,225,946,000
ocl_kernel1	13,381	9,368	1.428	4.77%	4.76%	41,026,146,000	28,722,288,000
ocl_kernel3	124	92	1.348	0.04%	0.05%	380,184,000	282,072,000

Double click the selected OpenCL* kernel to see the assembly source code and its relevant sampling information:



Address	Source	Pena	CPU_CLK	INST_RET
0x24D	pop ebp			
0x24E	ret			
0x2D7	...kernel2: push ebp			
0x2D8	mov ebp, esp			
0x2DA	and esp, -0x10			
0x2E0	push edi			
0x2E1	push esi		1	
0x2E2	sub esp, 0x8h			
0x2E5	mov eax, DWORD PTR [ebp+020h]			
0x2E8	mov ecx, DWORD PTR [ebp+024h]			
0x2EB	add ecx, DWORD PTR [eax]			
0x2ED	mov eax, ecx			
0x2EF	shl eax, 0x4h			
0x2F2	mov edx, eax			
0x2F4	add edx, DWORD PTR [ebp+0ch]			
0x2F7	add eax, DWORD PTR [ebp+08h]			
0x2FA	mov esi, 0x989680h			
0x2FF	mov edi, DWORD PTR [ebp+010h]			
0x302	...l2+0x2b: movaps xmm0, XMMWORD PTR [eax]	16,241	40,354	
0x305	mulps xmm0, XMMWORD PTR [edx]	3,600	65	
0x308	haddps xmm0, xmm0	732	156	
0x30C	haddps xmm0, xmm0	40,620	40,312	
0x310	movss DWORD PTR [edi+ecx*4], xmm0	66,194	12,654	
0x315	dec esi	6,013	174	
0x316	jnz \$-20 (000000000000002bh)	79		
0x31C	...l2+0x45: add esp, 0x8h			

Address	Size	Function	Class	CPU_CLK_UNHALTED.THREAD (86)	INST_RETIRED.ANY (86)	C:
-----	-----	--- Selected Range ---	-----			
0x6C	0x4E	ocl_kernel1		13,381	9,368	
0x2D7	0x4E	ocl_kernel2		133,480	93,715	
0x542	0x4E	ocl_kernel3		124	92	
0x7AD	0x4E	ocl_kernel4		133,288	93,681	

4.6 Using the Implicit CPU Vectorization Module

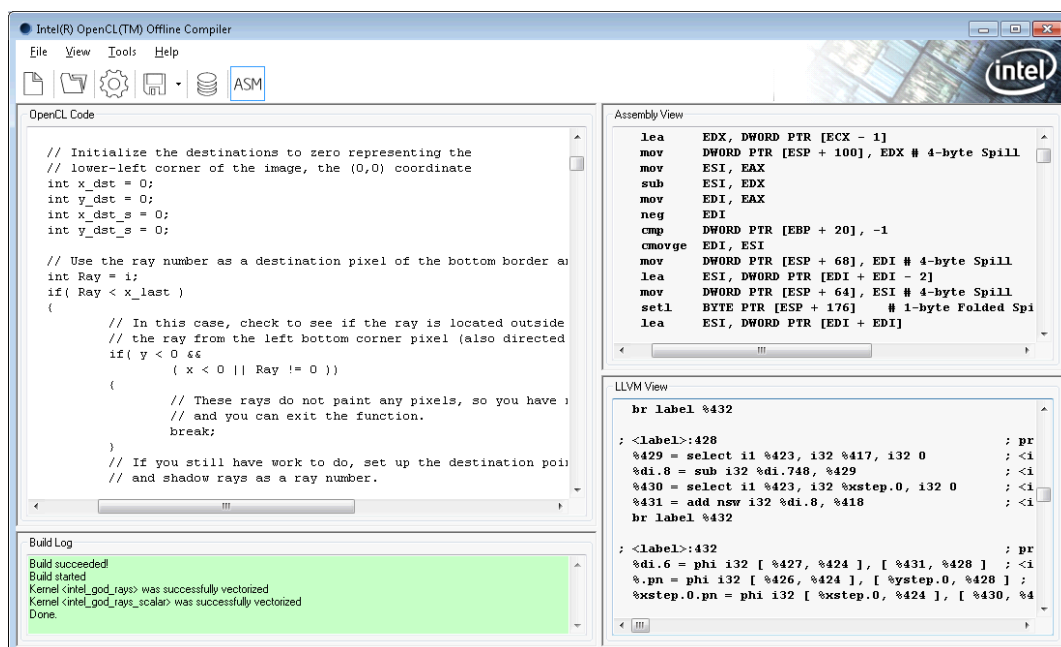
The Intel® OpenCL SDK uses an implicit CPU vectorization module as part of the program build process. This vectorization module aims to merge together the execution of several work items, using the Intel vector instruction set. For more information, see the document *Writing Optimal OpenCL* Code With Intel® OpenCL SDK*.

The vectorization module is applied by default and it assumes int/unit/float types are used within a kernel. If vector types are used, it is better to turn off the implicit CPU vectorization module, with the OpenCL* attribute `vec_type_hint`. Specifying `__attribute__((vec_type_hint(<typen>)))` for a kernel with any type other than int or float disables the implicit CPU vectorization module for that kernel.




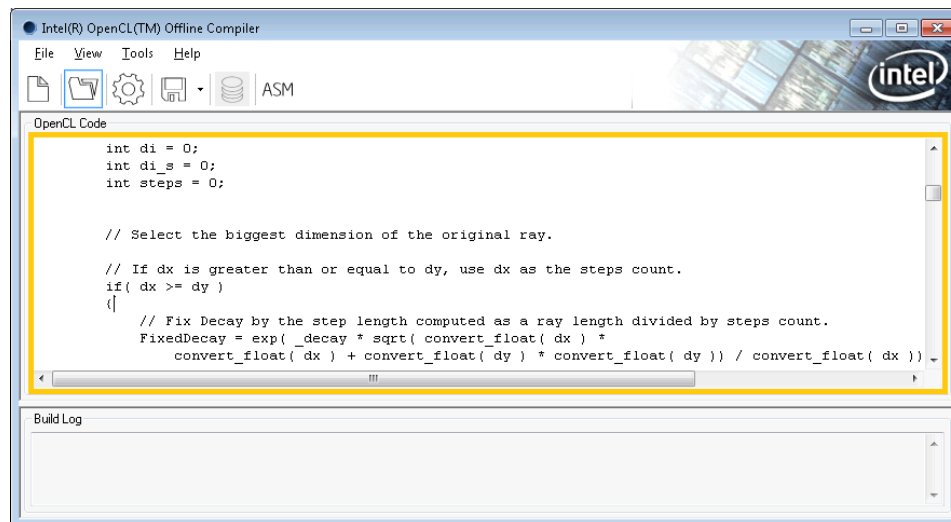
4.7 Using the Intel® OpenCL SDK Offline Compiler


The Intel® OpenCL SDK provides a unique tool that offers full offline OpenCL* language compilation, including an OpenCL* syntax checker, LLVM viewer, Assembly language viewer and intermediate program binaries generator.

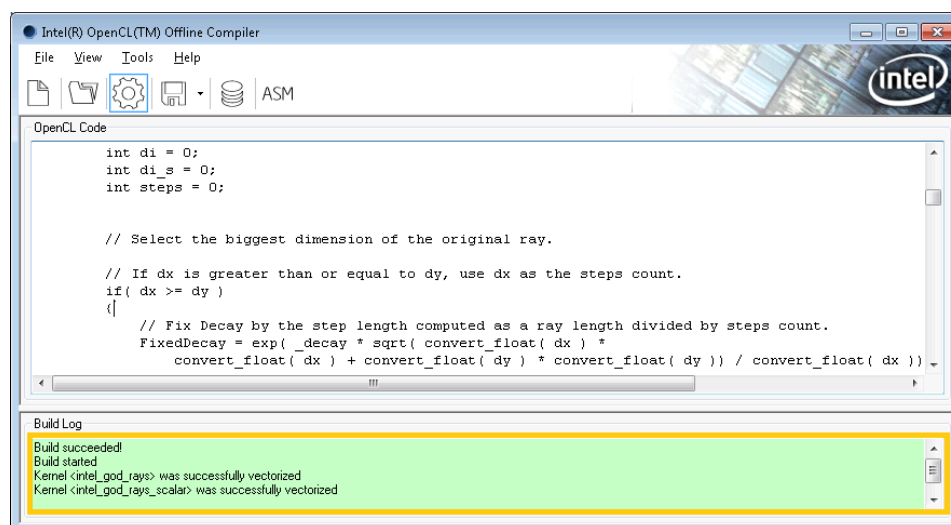


4.7.1 Building OpenCL* Kernels


Use the **OpenCL* Code** text box to write your OpenCL* code, or load the code from an external text file by clicking the **Load From File**  button on the tool-bar, or by selecting **File > Load From File....**



Click the **Build**  button on the commands tool-bar or select **Tools > Build** from the application's main menu to build the OpenCL* code. Look at the **Build Log** text box below to see information on the build status. If the build succeeds, the text box's background color will become green, otherwise it will become red.

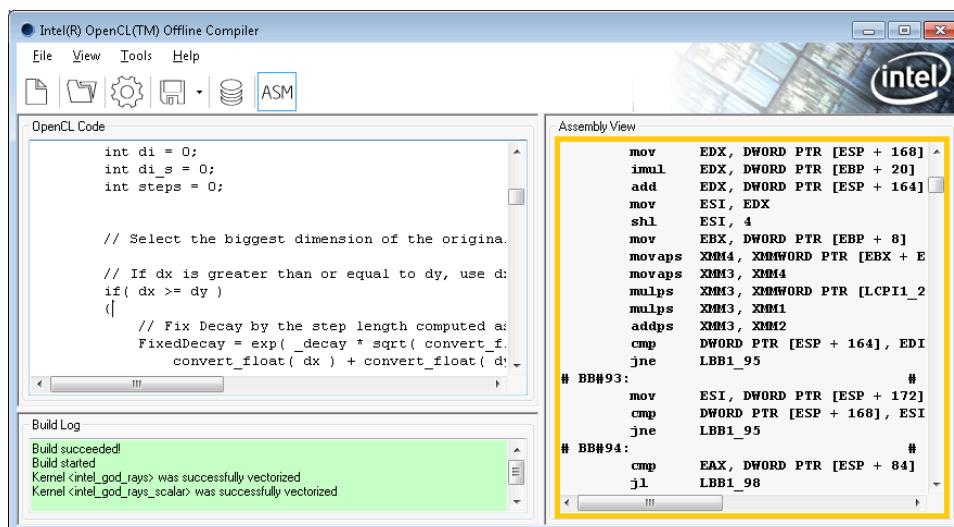


4.7.2 Viewing the Generated Assembly Code

The Intel® OpenCL SDK Offline Compiler tool enables you to see the generated assembly code of OpenCL* code. After the build successfully completes, click the **View Assembly Code**  button on the application's tool bar or select **View > Show**



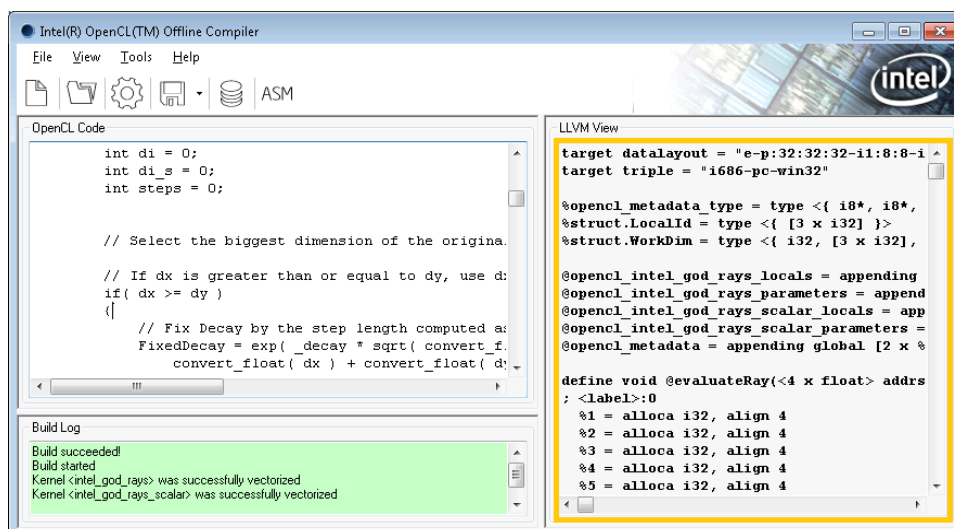
Assembly Code from the application's main menu. The assembly code appears in the **Assembly Code** text box, to the left of the **OpenCL* Code** text box.



Clicking again on the **View Assembly Code** button hides the **Assembly Code** text box.

4.7.3 Viewing the Generated LLVM Code


It is also possible to see the generated LLVM code of OpenCL* code. When the build successfully completes, select **View > Show LLVM Code**. The LLVM code appears in the **LLVM Code** text box, to the right of the **OpenCL* Code** text box.

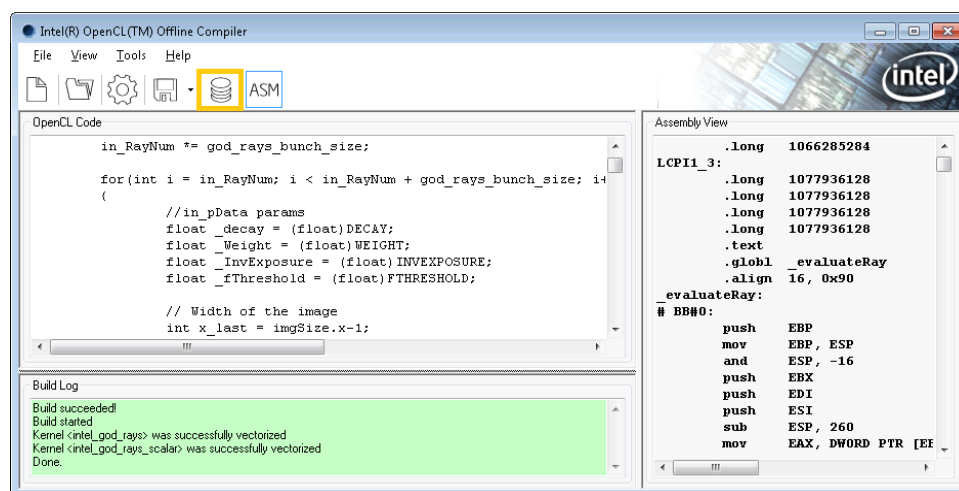




4.7.4 Generating Intermediate Program Binaries

The Intel® OpenCL SDK Offline Compiler tool enables you to generate program binaries of OpenCL* code. These binaries can be used later by the application, in order to create a program from binaries (`clCreateProgramFromBinary(...)`).


After the build successfully completes, click the **Create Program Binary**  button or select **Tools > Create Program Binary**.



Choose the location and the name of the program binary in the **Save As** dialog box and click **OK** to save the file.

4.7.5 Saving the Code to a Text File

The Intel® OpenCL SDK Offline Compiler tool enables you to save the generated Assembly/LLVM code, as well as the OpenCL* code that you entered.

Click the **Save As...**  button and select the type of code that you want to save (Assembly/LLVM/OpenCL*) or select **File > Save**.

4.7.6 Working with the Command Prompt

The Intel® OpenCL Offline Compiler enables you to run the tool from the command line. There are two version of the tool, a 32-bit version (`ioc32.exe`) and a 64-bit version (`ioc64.exe`). The command-line tool is located in the following directory:



`$(INTELOCLSDKROOT)\bin\x86` - for the 32-bit version of the tool

`$(INTELOCLSDKROOT)\bin\x64` - for the 64-bit version of the tool

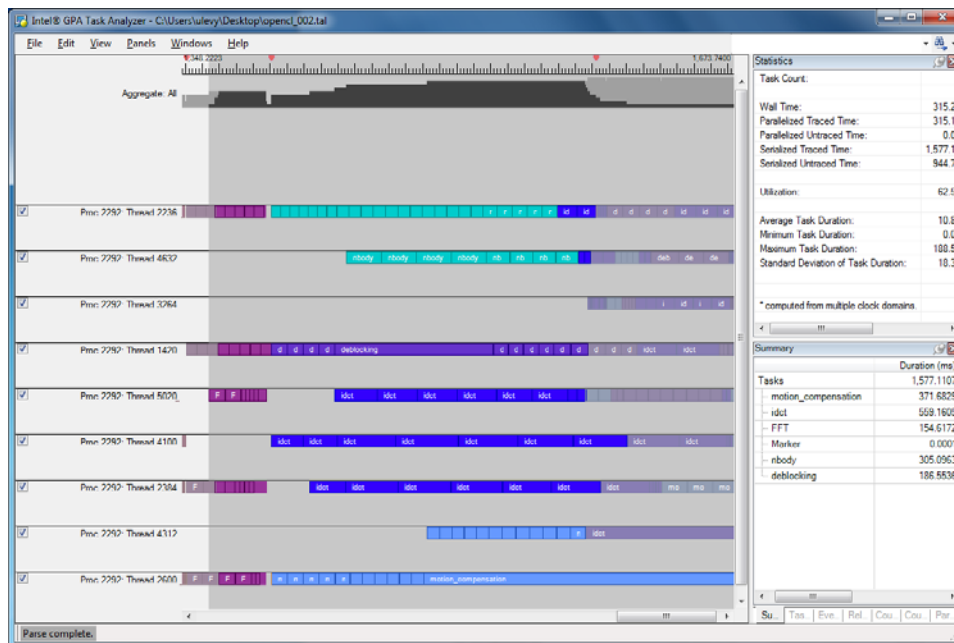
Use the command **`ioc -help`** to display descriptions of all available switches in the command window.

4.8 Tracing OpenCL* Commands with the Intel® GPA Task Analyzer

4.8.1 About Intel® GPA Task Analyzer

The Intel® GPA Task Analyzer visualizes the execution profile of the various tasks in your code over a period of time. The tool collects real-time trace data during the application run and provides a system-wide picture of the way your code executes on the various CPU and GPU cores in your system. The tool infrastructure automatically aligns clocks across all cores in the entire system so that you can analyze CPU-based workloads together with GPU-based workloads within a unified time domain.

For more information, see the **Intel GPA Task Analyzer Help**.



The Intel® OpenCL SDK provides built-in capturing support, which enables developers to see the distribution of their application's OpenCL* commands (kernels and memory operations) across the system's software threads. You can use the generated trace file later to analyze the application's flow (identify critical bottlenecks, etc.) and to improve its performance.

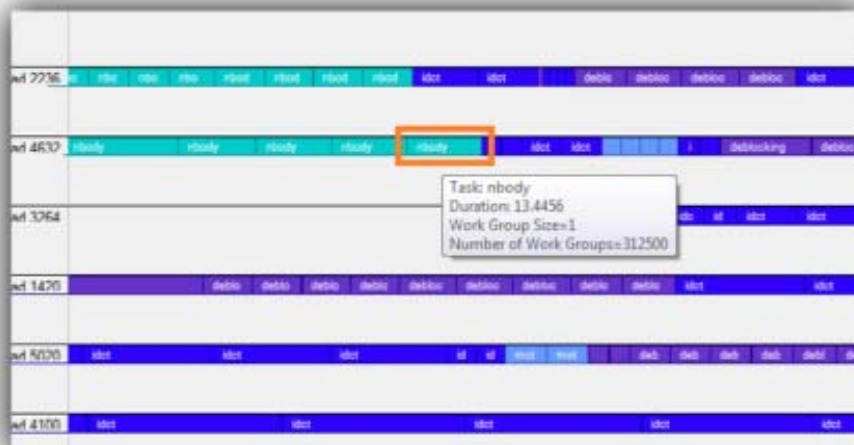
4.8.2 OpenCL* Trace Layout

After the trace data file has been created, you can load the generated OpenCL* trace data file into the Intel® GPA Task Analyzer application. When the trace file is loaded, the following items are display:

4.8.2.1 Tasks

There are two types of tasks that are available on the timeline tracker:

OpenCL* Kernels. Each task on the timeline tracker represents a cluster of workgroups that belongs to the same kernel. All tasks that belong to the same kernel will be colored in the same color; each kernel will have a different color.



Memory operations. In addition to OpenCL* kernels, there will be tasks on the timeline representing a memory operation (Read, Write, Copy, Map and Un-map). Read/Write/Copy commands are red, where Map and Unmap commands are maroon.

4.8.2.2 Markers

For each enqueue command, the following markers appear on the time ruler:

- A marker indicating the time command was entered to the commands queue
- A marker indicating the time command finished executing

4.8.3 Using the Intel® OpenCL SDK Tracer

The Intel® OpenCL SDK Tracer enables you to run an OpenCL* application in instrumentation mode and to automatically generate Task Analyzer trace files.

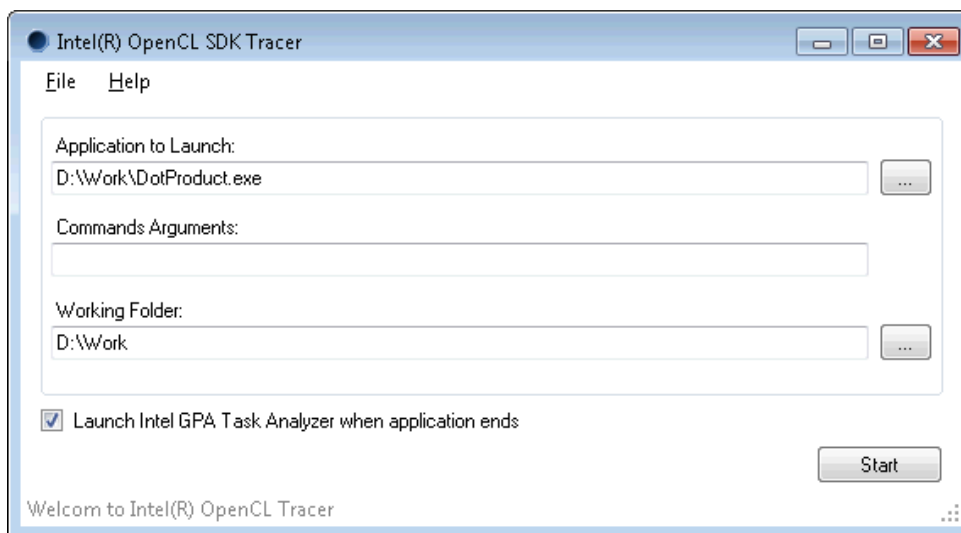
In the **Application to Launch** field, specify the name of the application or program that you want to run.

The **Working Folder** field is assigned automatically if you use the **Browse** button in the previous step. Change the default working directory, if required.

Enter any command line arguments pertinent to the application, if required.



Mark the **Launch Intel GPA Task Analyzer when application ends** check box if you would like to automatically launch the Task Analyzer application with the generated trace file when the application ends



Click **Start** to create the Activity and start trace generation. You can cancel the execution at any time by clicking the **Cancel** button.

4.8.4 Generating a Trace File Manually

To generate a trace file manually:

1. Set a new environment variable named **CL_CONFIG_USE_TASKALYZER** to the value **True** (setting the environment variable back to **False** disables the tracing)
2. Copy the Task Analyzer configuration file (tal.conf), located in **\$(INTELOCLSDKROOT)\tools\tal**, to the **working folder** of your application. You can customize the TAL configuration file's properties according to your needs.

4.8.5 Task Analyzer Configuration File

In order to generate a trace data file, the TAL configuration file needs to be located under the working folder of the application. It is recommended to use the default setting defined in the file; however, you can modify the TAL settings. This chapter outlines the list of configurable parameters in the TAL configuration file.



```
OUTPUT file <filename>
```

generates a TAL file named **<filename>**. **<filename>** may contain any or all of the special escapes:

- %p - Embeds process ID in the filename.
- %n - Embeds a unique ID in the filename s.t. no existing files are overwritten

```
OUTPUT net [port=<int>] [timeout=<int>
```

Instruct TAL to output to the network.

The optional **port=<portnum>** argument tells TAL to use a port other than the default one.

The optional **timeout=<int>** argument tells TAL how long to wait for a network connection.

- -1 means wait until a network connection is established.
- 0 means don't wait for a network connection to be established.
- >0 the number of milliseconds to wait for a connection.

```
MAX_MEM <max memory footprint in bytes>
```

Set the maximum size of the memory footprint, in bytes, that is allocated for the trace data.

```
LEVEL <level>
```

Set the TAL log level, where **<level>** is a number greater than or equal to 0, or "MAX" for TAL_LOG_LEVEL_MAX.

```
CATEGORIES <ALL, csv_delimited list of categories to enable>
```

Set TAL to capture only functions with a provided category. Each item in the csv-delimited list of categories must be one of the following:

- **<integer>**: Enables TAL_CATEGORY_<integer>. NOTE: This is the category number, not the category mask.



- **<string>** : Enables a category of a given name, as defined by `TAL_DescribeCategoryMask`.
- **ALL**: Enables all categories.

4.9 Working with the *cl-fast-relaxed-math* Flag

The `cl-fast-relaxed-math` flag sets the optimization options `-cl-finite-math-only` and `-cl-unsafe-math-optimizations`. This enables optimizations for floating-point arithmetic that may violate the IEEE 754 standard and the OpenCL* numerical compliance requirements defined in the specification. This option causes the preprocessor macro `__FAST_RELAXED_MATH__` to be defined in the OpenCL* program.

Intel® OpenCL SDK provides such optimizations to the **floating-point** version of the following built-in math functions:

- gentype **logb** (gentype *x*)
- int *n* **ilogb** (gentype *x*)
- gentype **fdim** (gentype *x*, gentype *y*)
- gentype **fmod** (gentype *x*, gentype *y*)
- gentype **hypot** (gentype *x*, gentype *y*)
- gentype **fract** (gentype *x*, `__global`/`__local`/`__private` gentype **iptr*)
- gentype **sqrt** (gentype)
- gentype **rsqrt** (gentype)
- gentype **fmax** (gentype *x*, gentype *y*)
- gentype **fmin** (gentype *x*, gentype *y*)



Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel® Compiler User and Reference Guides" under "Compiler Options." Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.