Application Performance Using Intel® OpenCL™ Driver Diagnostics Sample User's Guide

By Bartosz Dunajski (Intel) (https://software.intel.com/en-us/user/1272659), Updated July 20, 2016 Translate

Intel® SDK for OpenCL™ Applications - Samples

Download Code Samples (/sites/default/files/managed/7c/dc/driver-diagnostics-sample.7z)

Contents

Introduction

This article presents a driver diagnostics sample that demonstrates how to use the Intel® OpenCL™ extension to help get better performance results in applications. The article also discusses the most common mistakes made during application development and describes the diagnostic output (performance hint) feature that tells which behavior is more desirable to execute faster and shows measured time differences.

Motivation

Better application performance comes from code quality and knowledge. To help you improve your code, the OpenCL Driver provides a feature that helps you understand how your implementation impacts performance. We can find many examples showing how to write good code, but how can we check whether our implementation fits driver expectations? A returned hint points out which API call can cause performance drop and tells how to fix it. It works on selected context so we can use it as an automated bug finder.

How Driver Diagnostics Works

Types of hints

You can use three types of hints, depending on needs:

1. BAD
   This hint is provided when a specific execution can cause performance drop. A potential solution to the problem is also described. Example: Non-zero copy buffer.
2. NEUTRAL
   There is a potential performance impact, but it’s expected. This hint may be also used to inform the

user about specific actions.

3. GOOD
   In contrast to the BAD type of hint, this hint suggests the right approach. Example: zero copy buffer.

Each level is described as a flag that is passed as a value of the

CL_CONTEXT_SHOW_DIAGNOSTICS_INTEL parameter during the **clCreateContext** call:

CL_CONTEXT_DIAGNOSTICS_LEVEL_GOOD_INTEL

CL_CONTEXT_DIAGNOSTICS_LEVEL_BAD_INTEL

CL_CONTEXT_DIAGNOSTICS_LEVEL_NEUTRAL_INTEL

Receiving a hint

Information about the enabling feature and desired hint level must be passed as a parameter during the **clCreateContext** call. Driver output is provided by the **callback function** which is also passed to the same API call.

The example below shows how to create the proper context and receive the hint string.

```
void CL_CALLBACK NotifyFunction( const char * pErrInfo, const void *
pPrivateInfo, size_t size, void * pUserData )
{
    if( pErrInfo != NULL )
    {
        PrintHint( pErrInfo ); // Parse hint
    }
};

void SampleClass::SampleFcn( cl_device_id deviceID )
{
    cl_int ErrorCode;
    m_pfnNotify = NotifyFunction;

    m_hintLevel = CL_CONTEXT_DIAGNOSTICS_LEVEL_BAD_INTEL |
CL_CONTEXT_DIAGNOSTICS_LEVEL_NEUTRAL_INTEL;

    cl_context_properties properties[ ] =
    {
        CL_CONTEXT_SHOW_DIAGNOSTICS_INTEL,
(cl_context_properties)m_hintLevel,
        0
    };

```

Output

Each time the driver decides that a hint should be provided, the **NotifyFunction()** is executed and **pErrInfo** is the hint string. We should also remember mutual exclusion when operating on a string. Example: Inserting to a std container.

Examples for BAD and GOOD levels:

```
Performance hint: clCreateBuffer with pointer 3195c24 and size 3900
doesn't meet alignment restrictions. Size should be aligned to 4096
bytes and pointer should be aligned to 4096. Buffer is not sharing the
same physical memory with CPU.

Performance hint: clCreateBuffer with pointer 30d5000 and size 4096
meets alignment restrictions and buffer will share the same physical
memory with CPU.
```

As we can see, the provided hints show which API call is affected and how to fix the problem (for the BAD case). In most scenarios, a BAD hint has an opposite GOOD hint that confirms that the problem doesn't occur—but not always. Sometimes when a BAD hint disappears, we won't see any confirmation from a GOOD hint.

Sample Prerequisites

You must have the latest Intel® Graphics Driver supporting the *cl_intel_driver_diagnostics* extension for the OpenCL GPU Device (the device needs to support the extension) and Intel SDK for OpenCL installed.

The code provided is not OS-specific. To run the sample you need to prepare it using cmake (version >= 3.1.0). There is a CMakeLists.txt file in the sample directory:

```
> cmake CMakeLists.txt
```

Then you can use the generated files by CMake* to build a sample (for example, using Microsoft Visual Studio*).

Sample Output

We can select up to three test scenarios to execute: buffers, kernels, enqueues. Each scenario has its own test set. And each test presents similar output:

```
Testing bad case…
[Received hints]

Testing good case…
[Received hints]

Result: Bad case [x] us – Good case [y] us – BAD/GOOD [x]/[y]
```

The sample shows two opposite cases where we can see potential performance gain. Time is measured before the affected call and after **clFinish()** to measure the actual work.

Controlling The Sample

Since this feature is Intel-specific for the GPU driver you don't need to select the platform or device manually.

The sample executable is a console application. Use the following command-line arguments for the sample control:

Option	Description
-h, -help	Show help text and exit.
-test_scenario_buffers	Execute "buffers" scenario.
-test_scenario_kernels	Execute "kernels" scenario.
-test_scenario_enqueues	Execute "enqueues" scenario.
-print_all_hints	Print all received hints if used. Otherwise only expected hints for specific case will be shown.
-print_descriptions	Print description for currently executing test case if set.

APIs Used

This sample uses the following APIs:

- clBuildProgram
- clCreateBuffer
- clCreateCommandQueue
- clCreateContext
- clCreateKernel
- clCreateProgramWithSource
- clEnqueueMapBuffer
- clEnqueueNDRangeKernel
- clEnqueueReadBuffer
- clEnqueueUnmapMemObject
- clEnqueueWriteBuffer
- clFinish
- clGetDeviceIDs

- clGetDeviceInfo
- clGetPlatformIDs
- clGetPlatformInfo
- clReleaseCommandQueue
- clReleaseContext
- clReleaseKernel
- clReleaseMemObject
- clReleaseProgram
- clSetKernelArg

References

- OpenCL Khronos Registry (https://www.khronos.org/registry/cl/) and clCreateContext (https://www.khronos.org/registry/cl/sdk/1.0/docs/man/xhtml/clCreateContext.html)
- Intel SDK for OpenCL* Applications – Optimization Guide (/sites/landingpage/opencl/optimization-guide/)
- Intel SDK for OpenCL* (/en-us/intel-opencl)
- CMake (https://cmake.org/) – official website