

High Dynamic Range Tone Mapping Post Processing Effect Multi-Device Version

Sample User's Guide

*Intel® SDK for OpenCL * Applications - Samples*

Document Number: 329765-005US

Contents

Contents	2
Legal Information	3
About Multi-Device Version of the Tone Mapping Sample	4
Introduction	4
Motivation	4
Right Way to Study the Sample	4
Creating Shared Context	5
Work Distribution Primer	5
Efficient Resource Sharing	5
Writing to a Shared Resource	6
Synchronization Caveat	6
Updating Input Data in Paralell with Processing	7
Understanding OpenCL* Performance Characteristics	7
APIs Used	7
Reference (Native) Implementation	8
Controlling the Sample	8
References	8

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.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

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/.

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 and the OpenCL logo are trademarks of Apple Inc. used by permission from Khronos.

Microsoft product screen shot(s) reprinted with permission from Microsoft Corporation.

Copyright © 2010-2013 Intel Corporation. All rights reserved.

Optimization Notice
<p>Intel's compilers 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 SSE2, SSE3, and 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. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.</p> <p>Notice revision #20110804</p>

About Multi-Device Version of the Tone Mapping Sample

Original Tone Mapping SDK sample demonstrates how to use high dynamic range (HDR) rendering with tone mapping effect in OpenCL*. Please refer to the sample description for the details on the technique itself.

This version features multi-device support, specifically the simultaneous usage of CPU and Intel® Processor Graphics OpenCL devices. It also shows some basic approach to the load-balancing, resources sharing, and so on.

Introduction

Refer to the original Tone-Mapping sample for details on the high dynamic range and its applications. In the context of this sample the original post-processing technique itself is used for merely illustrative purposes, while the main focus is simultaneous usage of both CPU and Intel Processor Graphics OpenCL devices.

Intel OpenCL implementation enables interfacing with Intel Processor Graphics and CPU devices by use of common runtime.

It is also possible to create a “shared” context with both devices to enable resource sharing. Still the OpenCL specification does not assume any “shared queue”, so an application should perform job distribution for the devices.

This sample is an illustration of the basic principles for working with both CPU and Intel Processor Graphics OpenCL devices simultaneously.

Motivation

Running simultaneously on both CPU and Intel Processor Graphics device requires proper synchronization for example with respect to shared resources. It is also important to follow certain recommendations to get true resource sharing between devices. The sample shows basic approaches to both tasks.

Simple way of static work assignment for the devices might result in lower overall performance. The sample shows simple strategy for adaptive partition the work between the devices (command queues).

General optimization strategy for a heterogeneous application is to maximize the utilization of the underlying devices by keeping them busy most of the time. It is important to hide data transfer latencies by use of asynchronous transfers. The sample implements basic double-buffering technique.

Right Way to Study the Sample

This sample is a heterogeneous example of Tone Mapping, which requires understanding of OpenCL programming for single-device case. Refer to the original Tone Mapping sample for example of the single-device case.

The sample supports running single-device case (actually this is the default mode). So it is recommended to run sample on CPU device first, then on Intel HD Graphics, and finally on both devices simultaneously. Refer to the “Controlling the sample” section for details on the respective command-line switches. You may also want to capture the resulted performance of those three cases to compare the results.

Running only CPU or Intel HD Graphics device cases is conceptually identical and similar to the original Tone-Mapping sample, while the case of a shared context is different. Step through the application in debugger to understand its basic logic, which means creating a shared context, resources, and other

initialization routine, splitting the job between the devices, firing the tasks and synchronization for results. Refer to the `ExecuteToneMappingKernelSimple` function which is good starting point. You can map these stages to the chapters of this document.

Sample also has a mode when it emulates the case when new input data to process actually arrives each frame. The two of possible ways to update the input buffer are presented in the `ExecuteToneMappingKernelWithInputFrameUpdate` function.

Creating Shared Context

The sample supports shared context for CPU and Intel Processor Graphics devices.

The sample uses explicit device list when creating a (shared) context:

```
cl_device_type dev =
g_bUseBothDevices ? (CL_DEVICE_TYPE_CPU+CL_DEVICE_TYPE_GPU)
                  : (g_bRunOnPG ?CL_DEVICE_TYPE_GPU:CL_DEVICE_TYPE_CPU);
g_context = clCreateContextFromType(props, dev, NULL, NULL, NULL);
```

In case of single-device context, this enables the common runtime to use faster/direct way of communications. So this is a more performance-friendly way than creating context with `CL_DEVICE_TYPE_ALL` while you actually need just one device.

Notice that shared context does not imply any “shared queue”, so you still need to create a separate queue per device. The next section describes how work is distributed between these command queues.

Work Distribution Primer

A proper strategy is allocating work according to the *current* load and speed of devices. A device speed can be affected by OS or driver scheduling decisions and also dynamic frequency scaling (see section Efficient Resource Sharing).

The sample implements the “intra-frame” approach to the dynamic scheduling. Specifically, the input frame is being split between devices. The actual splitting ratio is adjusted dynamically, based on how fast the devices complete the tasks. Refer to the `ComputeSplittingRatio` function in the `ToneMappingMultiDevice.cpp`.

Below is an example of output from running simultaneously on CPU and Processor devices (“-m” command-line option, refer to the “Controlling the Sample” section). Notice how initial value of job splitting ratio (50% by 50%) quickly adapts to the speed of devices. So that starting 3rd frame the execution time of each device is approximately the same, which is balanced:

```
Execution time: for #1 device is 41.877 ms, for the #2 device: 18.106 ms
The frame was splitted as 50.0% by 50.0% between device #1 and #2
Execution time: for #1 device is 24.481 ms, for the #2 device: 21.105 ms
The frame was splitted as 40.1% by 59.9% between device #1 and #2
Execution time: for #1 device is 22.900 ms, for the #2 device: 22.441 ms
The frame was splitted as 38.9% by 61.1% between device #1 and #2
Execution time: for #1 device is 23.282 ms, for the #2 device: 22.902 ms
The frame was splitted as 38.8% by 61.2% between device #1 and #2
Execution time: for #1 device is 22.782 ms, for the #2 device: 22.266 ms
The frame was splitted as 38.7% by 61.3% between device #1 and #2
```

Efficient Resource Sharing

Objects allocated at the context-level are shared between devices in the shared context. Thus buffers created with regular `clCreateBuffer` are effectively shared by default, so no implicit copying or synchronization happens between devices.

The memory referenced with `CL_MEM_USE_HOST_PTR` should be properly aligned (refer to the Optimization Guide for more information):

```
cl_uint min_align, min_align_cpu, min_align_gpu;
```

```

clGetDeviceInfo(cpu_device_ID, CL_DEVICE_MEM_BASE_ADDR_ALIGN,
               sizeof(cl_uint), &min_align_cpu, NULL);
clGetDeviceInfo(gpu_device_ID, CL_DEVICE_MEM_BASE_ADDR_ALIGN,
               sizeof(cl_uint), &min_align_gpu, NULL);
min_align = max(min_align_cpu, min_align_gpu);
g_min_align /= 8; //in bytes

cl_float* inputArray = (cl_float*)_aligned_malloc(iMemSize, min_align);
cl_buffer bufferShared=
    clCreateBuffer(g_context, CL_MEM_READ_ONLY|CL_MEM_USE_HOST_PTR,
                  sizeof(CHDRData), pData, NULL);

```

Writing to a Shared Resource

According to OpenCL* spec you need to ensure that commands that change the content of a shared memory object complete in the previous command queue before the memory object is used by commands executing in another command-queue.

The only way to write data (or output kernel results) to the same buffer *simultaneously* with two devices is using (non-overlapping) sub-buffers. The sub-buffers should be properly aligned.

```

//make sure that cpuPortion is properly aligned first!
cl_buffer_region cpuBufferRegion = { 0, cpuPortion};
cl_buffer_region gpuBufferRegion = { cpuPortion, theRest};

cl_buffer subbufferCPU = clCreateSubBuffer(bufferShared, 0,
    CL_BUFFER_CREATE_TYPE_REGION, &cpuBufferRegion, &err);
cl_buffer subbufferGPU = clCreateSubBuffer(bufferShared, 0,
    CL_BUFFER_CREATE_TYPE_REGION, &gpuBufferRegion, &err);
//now work with 2 sub-buffers simulteneoulsy
..
//the sub-resources should be released properly
clReleaseMemObject(subbufferCPU);
clReleaseMemObject(subbufferGPU);
clReleaseMemObject(bufferShared);

```

Refer to the PrepareSubResources function in the ToneMappingMultiDevice.cpp.

Synchronization Caveat

Notice that after submitting commands to the respective (regular in-order) device queues it is important to flush both queues to ensure processing is started in parallel.

If you block with `clWaitForEvents` or `clFinish` on the first queue without submitting or flushing commands for the second queue, the things will be serialized. The reason is that program flow does not resume until `clWaitForEvents` or `clFinish` is returned.

So the right way is to submit or flush commands to both queues and then wait on event objects associated with "CPU" and "GPU" queue (error checking is omitted):

```

cl_event eventObjects[2];

//notice that kernel object itself can be the same (shared)
clEnqueueNDRRangeKernel(gpu_queue, kernel, ... &eventObjects[0]);
//other commands for the GPU queue
//...
//now let's flush first queue
//notice it is NOT clFinish or clWaitForEvents to avoid serialization
clFlush(gpu_queue);

clEnqueueNDRRangeKernel(cpu_queue, kernel, ... &eventObjects[1]);
//other commands for the CPU queue
//...
//flushing queue to make things rolling on GPU in parallel to CPU
clFlush(cpu_queue);

```

```
//now when both queues are flushed let's wait for both kernels to complete  
clWaitForEvents(2, eventObjects);
```

Updating Input Data in Paralell with Processing

Sample has a mode (refer to the “Controlling the Sample” section) when it emulates the case when the new input data to process actually arrives in each frame. Sample implements two ways to update the input buffer:

- Synchronous mode, when the buffer is processed and just then refilled.
- Double-buffering scheme. In this approach, there are two buffers, and while the first is processed, the second is updated, and then they are swapped.

It is important to perform updating in parallel with processing. Refer to the `ExecuteToneMappingKernelWithInputFrameUpdate` that actually features both ways.

Understanding OpenCL* Performance Characteristics

The actual benefit of using both devices depends on several factors:

- Relative speed of underlying devices.
- Work granularity - the grain size must be large enough to amortize associated overheads from additional scheduling and kernel submission.

Devices influence the speed of each other through dynamic frequency scaling and contention for shared system resources like (global) memory bandwidth.

Hide data transfer latencies, and compare performance of sync and aync input buffer update modes on your machine.

The sample measures and outputs the complete time to process `TEST_RUNS` frames. This includes time to create (sub-)buffers, various kernel setup routine, even printing statistics. Thus these overheads drag down the speedup from using both devices. The “pure” speedup of the kernel execution is more close to theoretical peak, but it is really important to track the overall time to avoid introducing any inefficiencies (for example, related to additional synchronization and/or resource sharing). The bottom-line is that experimenting in the context of the real heterogeneous application is strongly recommended.

Finally, application-level effects like interoperability with other APIs (like DirectX*) might introduce additional synchronization dependencies. Refer to the Optimization Guide for optimization guidelines on avoiding various penalties.

APIs Used

This sample uses the following APIs:

- `clCreateKernel`
- `clCreateContextFromType`
- `clGetContextInfo`
- `clCreateCommandQueue`
- `clCreateProgramWithSource`
- `clBuildProgram`
- `clCreateBuffer`
- `clSetKernelArg`
- `clEnqueueNDRangeKernel`
- `clEnqueueReadBuffer`
- `clReleaseMemObject`
- `clReleaseKernel`
- `clReleaseProgram`

- `clReleaseCommandQueue`
- `clReleaseContext.`
- `clCreateSubBuffer`
- `clFlush`
- `clGetEventProfilingInfo`
- `clEnqueueMapBuffer`
- `clEnqueueUnmapMemObject`
- `clFinish`
- `clWaitForEvents`

Reference (Native) Implementation

Reference implementation is done in `ExecuteToneMappingReference()` routine of `ToneMappingMultiDevice.cpp` file. This is single-threaded code that performs exactly the same tone mapping effect sequence as the OpenCL implementation, but using conventional nested loop in C with SSE optimizations. Native kernel `EvaluateRaw()` that processes input HDR image is located in `ToneMappingNative.cpp`.

Controlling the Sample

The sample executable is a console application. Use following command line argument to run sample on Intel Processor Graphics.

Option	Description
<code>-h, --help</code>	Show this help text and exit.
<code>-p, --platform number-or-string</code>	Select platform, devices of which are used.
<code>-t, --type cpu gpu cpu+gpu</code>	Selects the device by type on which the OpenCL kernel is executed.
<code>--iteration-statistic</code>	Print per iteration statistics.
<code>--input-swapping</code>	Updating the input frame on each iteration, and compares sync./async. update mode.

References

- [1] <http://www.openexr.com/using.html>
- [2] "Fast Tone Mapping for High Dynamic Range Images" by Jiang Duan and Guoping Qiu
- [3] [Intel SDK for OpenCL Applications - Optimization Guide](#)