

1 Overview

Note that this sample is essentially the same as the TransferOverlapCPP sample, except that it uses the OpenCL C API; the TransferOverlapCPP sample uses C++ bindings for the OpenCL API.

1.1 Location `$<APPSDKSamplesInstallPath>\samples\opencl\cl\`

1.2 How to Run See the *Getting Started* guide for how to build samples. You first must compile the sample.

Use the command line to change to the directory where the executable is located. The pre-compiled sample executable is at `$<APPSDKSamplesInstallPath>\samples\opencl\bin\x86\` for 32-bit builds, and `$<APPSDKSamplesInstallPath>\samples\opencl\bin\x86_64\` for 64-bit builds.

Type the following command(s).

1. `TransferOverlap`
This runs the program with the following default options: `-i 1, -k 10, -x 1048576 (1MB), -s 0, -w 7, -l 0`.
2. `TransferOverlap -h`
This prints the help file.

1.3 Command Line Options Table 1 lists, and briefly describes, the command line options.

Table 1 Command Line Options

Short Form	Long Form	Description
-h	--help	Shows all command options and their respective meaning.
	--device	Devices on which the program is to be run. Acceptable values are <code>cpu</code> or <code>gpu</code> .
-q	--quiet	Quiet mode. Suppresses all text output.
-e	--verify	Verify results against reference implementation.
-t	--timing	Print timing.
	--dump	Dump binary image for all devices.
	--load	Load binary image and execute on device.
-d	--deviceId	Select deviceId to be used (0 to N-1, where N is the number of available devices).
	--flags	Specify compiler flags to build the kernel.
-p	--platformId	Select platformId to be used (0 to N-1, where N is the number of available platform).

Short Form	Long Form	Description
-v	--version	AMD APP SDK version string.
-x	--size	Size in bytes.
-i	--iterations	Number of timing loops.
-s	--skip	Skip the first n iterations for average.
-k	--kernelLoops	Number of loops in the kernel.
-w	--wavefronts	Number of wavefronts per compute unit.
-I	--inMemFlag	Memory flags for the input buffer. 0 CL_MEM_READ_ONLY 1 CL_MEM_WRITE_ONLY 2 CL_MEM_READ_WRITE 3 CL_MEM_ALLOC_HOST_PTR 4 CL_MEM_USE_PERSISTENT_MEM_AMD
-n	--noOverlap	Do not overlap <code>memset()</code> with kernel.
-l	--log	Prints complete timing log.

2 Introduction

This sample shows how to overlap the CL buffer transfer with running a device kernel; this is done using a zero copy `CL_MEM_USE_PERSISTENT_MEM_AMD` buffer and an in-order queue.

The CPU performs the data transfer (instead of the DMA engine or the GPU), freeing the GPU to perform compute.

The GPU kernel writes the result directly into host memory using another zero copy buffer of type `CL_MEM_ALLOC_HOST_PTR`.

The AMD APP SDK and driver must support the zero copy feature; otherwise the zero copy buffer allocation can fail or default to a non-zero copy buffer.

3 Overlap Using Zero Copy

The definition of a zero copy buffer is that its contents are not copied unless explicitly requested by the user (for example by using `clCopyBuffers`).

A `CL_MEM_USE_PERSISTENT_MEM_AMD` buffer is a zero copy buffer located on the device. The host can directly access it by using `memset()` or `memcpy()`. This is independent of GPU kernel execution. Calls to `cl*Map*()` and `cl*Unmap*()` for a zero copy buffer are typically very low cost since they do not result in any data transfer.

The code uses a double buffering set-up. The basic execution sequence is as follows:

```
map buffer 1
while(..)
{
    memset to buffer 1, overlapping with kernel 2
    unmap buffer 1

    map buffer 2
    launch kernel for buffer 1

    memset to buffer 2, overlapping with kernel 1
    unmap buffer 2
```

```

    map buffer 1
    launch kernel for buffer 2
}

```

It is important to initiate the map of one buffer before the kernel is launched on the other buffer; otherwise, the map finishes after the preceding kernel is completed, as required by in-order queue semantics.

4 Implementation Details

The overall test time and average loop time are printed out at the end of the test. To discount one-time startup costs (lazy allocation, etc.), the first iterations of the test loop are not included in the average overall runs.

1. Run `TransferOverlap -I 0 -n`. This uses normal device buffers, and the kernel calls are forced to finish before any other CL calls. All of `memset`, the DMA transfer, and the kernel are executed serially.
2. Run `TransferOverlap -I 0`. Using normal device buffers, the `memset` operation overlaps with the kernel execution, but the DMA still runs serially with the kernel.
3. Running `TransferOverlap` without options overlaps a `memset` into a `CL_MEM_USE_PERSISTENT_MEM_AMD` buffer with the kernel computation. If the kernel runtime is equal to the `memset` runtime, almost complete overlap can be achieved.

The kernel runtime can be tuned on a given platform using the `-k <n>` command line option. Use `TransferOverlap -n -k <n>` to obtain the actual front-to-back kernel execution time, and compare it to the time reported for `memset()` into the `CL_MEM_USE_PERSISTENT_MEM_AMD` buffer.

5 Example

The following is an example with actual numbers, using an AMD Radeon™ HD 5870 graphics card.

- `memset()` of 10 MB buffer in host memory: .001s at 10 GB/s
- DMA transfer of 10 MB buffer: .002 s at 5 Gb/s
- `memset()` of 10 MB buffer of
type `CL_MEM_USE_PERSISTENT_MEM_AMD`: .0033 s at 3 GB/s
- Kernel run time (tuned for optimal overlap): .0033 s

1. Serial case: `memset()` into host buffer + DMA +kernel, all run serially.
Total time = .001 s + .002 s + .0033 s = .0063 s
2. Partly overlapped case, `memset()` and kernel overlap, DMA is serial:
Total time = max(.001 s, .0033 s) + .002 s = .0053 s
3. Fully overlapped case, `memset()` into persistent and kernel overlap, no DMA:
Total time = max(.0033 s, .0033 s) = .0033 s

Contact

Advanced Micro Devices, Inc.
One AMD Place
P.O. Box 3453
Sunnyvale, CA, 94088-3453
Phone: +1.408.749.4000

For AMD Accelerated Parallel Processing:
URL: developer.amd.com/appsdk
Developing: developer.amd.com/
Support: developer.amd.com/appsdksupport
Forum: developer.amd.com/openglforum



The contents of this document are provided in connection with Advanced Micro Devices, Inc. ("AMD") products. AMD makes no representations or warranties with respect to the accuracy or completeness of the contents of this publication and reserves the right to make changes to specifications and product descriptions at any time without notice. The information contained herein may be of a preliminary or advance nature and is subject to change without notice. No license, whether express, implied, arising by estoppel or otherwise, to any intellectual property rights is granted by this publication. Except as set forth in AMD's Standard Terms and Conditions of Sale, AMD assumes no liability whatsoever, and disclaims any express or implied warranty, relating to its products including, but not limited to, the implied warranty of merchantability, fitness for a particular purpose, or infringement of any intellectual property right.

AMD's products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD's product could create a situation where personal injury, death, or severe property or environmental damage may occur. AMD reserves the right to discontinue or make changes to its products at any time without notice.

Copyright and Trademarks

© 2013 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, ATI, the ATI logo, Radeon, FireStream, and combinations thereof are trademarks of Advanced Micro Devices, Inc. OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos. Other names are for informational purposes only and may be trademarks of their respective owners.