AMD Accelerated Parallel Processing TECHNOLOGY

SAMPLE

Asynchronous Data Transfer

1 Overview

1.1 Location

$<*APPSDKSamplesInstallPath*>\samples\opencl\cpp_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. `AsyncDataTransfer`
   This command runs the program with the default options.
2. `AsyncDataTransfer -h`
   This command 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 meanings.
	--device	Devices on which the program is to be run. Acceptable values are `cpu` or `gpu`.
-q	--quiet	Quiet mode. Suppresses all text output.
-e	--verify	Verify results against reference implementation.
-t	--timing	Print timing-related statistics.
	--dump	Dump binary image for all devices.
	--load	Load binary image and execute on device.
	--flags	Specify compiler flags to build the kernel with.
-p	--platformId	Select platformId to be used (0 to N-1, where N is the number of available platforms).
-v	--version	AMD APP SDK version string.
-d	--deviceId	Select deviceId to be used (0 to N-1, where N is the number of available devices).
-i	--iterations	Number of iterations.
-x	--size	Size of the input buffer per kernel (in Bytes).
-k	--kernels	Number of kernels for execution.

2 Introduction

The Asynchronous Data Transfer sample demonstrates how to harness asynchronous memory transfer in OpenCL. The sample shows the overlap of kernel execution and data transfer in a device in which multiple same or different kernels are executing. The program workflow consists of three steps:

1. Data transfer (input) from Host to Device
2. Kernel Execution
3. Data transfer (output) from Device to Host

The data transfer step is interleaved with kernel execution step, thus improving overall performance.

The sample includes a sequential (synchronous) implementation in which the kernel execution and data transfer steps are invoked sequentially. The performance of the sequential version is compared with that of the Asynchronous version.

This sample is similar to the TransferOverlap sample. The difference is that the TransferOverlap sample used a single queue for data transfer and kernel execution, while this sample uses separate command queues for data transfer and kernel execution.

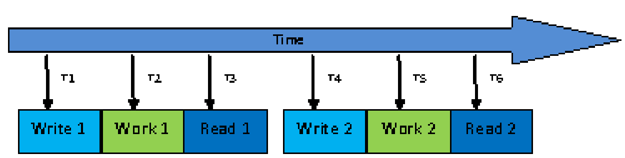
3 Implementation

The sample implements two versions of kernel executions:

1. Sequential (synchronous)
2. Asynchronous

3.1 Sequential (synchronous) execution implementation

As the name says, the kernels and data transfers run sequentially (writes-> executions-> reads). To make the operations sequential, one can either use blocking operations (i.e. each call returns after the operation is finished) or attach a wait event list. In this sample, blocking calls are used. The synchronous data transfer scenario is shown below:



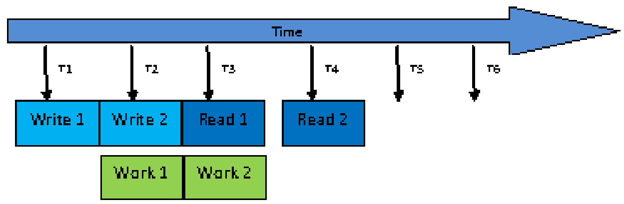
The following operations are executed in the order given. T1..T6 represent time instances.

1. T1: Write 1 - Writes data from HOST to DEVICE corresponding to Kernel 1.
2. T2: Work 1 - Kernel-1 executes.
3. T3: Read 1 - Reads data from DEVICE to HOST corresponding to Kernel 1.
4. T4: Write 2 - Writes data from HOST to DEVICE corresponding to Kernel 2.

5. T5: Work 2 - Kernel-2 executes.
6. T6: Read 2 - Reads data from DEVICE to HOST corresponding to Kernel 2.

3.2 Asynchronous execution implementation

In this version, non-blocking operations (write, read, execution) are used for making sure that the operations are asynchronous. Since the operations on each individual kernel are sequential (i.e. first write, then execute, and then read), wait and finish events are attached on each operation. The asynchronous data transfer scenario is shown below:



The following operations are executed in the order given.

1. T1: Work 1 - Writes data from HOST to Device corresponding to kernel 1.
2. T2: Kernel-1 execution starts. This overlaps with operation Write 2, viz. data transfer from HOST to Device for Kernel 2.
3. T3: While Read 1 operation starts for Kernel 1, Kernel-2 executes overlaps.
4. T4: Read 2 operation for Kernel 2.

The snapshot below shows the output of the Application Trace profile done using CodeXL. The overlapping bars show the asynchronous nature of the data transfer and kernel execution.

While Queue 0 is busy in kernel execution, Queue 1 and Queue 2 are concurrently performing read and write data transfers.

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/openclforum