AMD **Accelerated** Parallel Processing TECHNOLOGY

1 Overview

Note that this sample is essentially the same as the DeviceFission11Ext sample, except that this sample uses the OpenCL 1.2 device fission functionality; the DeviceFission11Ext sample uses OpenCL 1.1 and the `cl_ext_device_fission` extension.

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 default executables are placed in $<*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. `DeviceFission`
    This tests the kernel execution on multi-devices asynchronously.

2. `DeviceFission -h`
    This prints the help message.

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.
-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.
	--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 platforms).
-v	--version	AMD APP SDK version string.
-x	--length	Length of the input array.
-d	--deviceId	Select deviceId to be used (0 to N-1, where N is the number of available devices).

2 Introduction

The Device Fission functionality is supported only on the CPU.

This sample must be run in the OpenCL 1.2 environment. The following APIs are part of OpenCL 1.2:

`clCreateSubDevices`: Creates an array of sub-devices that each reference a non-intersecting set of compute units within a GPU, according to a partition scheme given by the API parameter properties.

`clEnqueueMigrateMemObjects`: Enqueues a command to indicate the device with which a set of memory objects is to be associated.

In this sample, a CPU device is partitioned into two sub-devices by using `clCreateSubDevices` and `CL_DEVICE_PARTITION_BY_COUNTS` property. Only one input buffer is created. Only one of the sub-devices is in charge of writing data into the input buffer; then, two sub-devices execute the kernels using data from that input buffer. The two sub-devices call their own kernels ("add" and "sub") on the input buffer and update their output buffers.