AMD **Accelerated** Parallel Processing TECHNOLOGY

SAMPLE

Sync vs Async Array Copy

1 Overview

1.1 Location

`$<APPSDKSamplesInstallPath>\samples\C++Amp\`

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 `$<APPSDKSamplesInstallPath>\samples\C++Amp\bin\x86\` for 32-bit builds, and `$<APPSDKSamplesInstallPath>\samples\C++Amp\bin\x86_64\` for 64-bit builds.

Type the following command(s).

1. `SyncVsAsyncArrayCopy`

   This runs the program with the default options: s = (1024 * 768 * 30).

2. `SyncVsAsyncArrayCopy -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`	Show all command options and their respective meaning.
`-q`	`--quiet`	Quiet mode. Suppresses text output.
`-e`	`--verify`	Verify results against reference implementation.
`-t`	`--timing`	Print timing.
`-d`	`--deviceId`	Select deviceId to be used (0 to N-1, where N is the ID of the device to be used).
`-v`	`--version`	AMD APP SDK version string.
`-x`	`--samples`	Number of example input values (multiples of three).

2 Introduction

When choosing between array and array_view in C++ AMP, arrays can be useful:

- Because DX interop APIs expect arrays as parameters.
- As staging buffers to optimize frequent data transfers that take place between the CPU and the GPU.
- To measure the performance of the data transfer to an accelerator.

- To copy to an accelerator asynchronously.

Array_views are useful for future-proofing code.

In this example, we explore staging buffers and how data transfers can be optimized using AMP arrays with them.

3 Implementation Details

While synchronously transferring large data sizes to, and from, an accelerator, the copy operation consumes much time. C++ AMP provides a set of global asynchronous `concurrency:copy_aync` functions that allow the current thread to perform some other task in the host application while the copy operation is in progress. This allows both the operations to be performed in parallel, instead of the host application waiting on the completion of the copy operation before proceeding. C++ AMP supports asynchronous `concurrency:copy_aync` functions corresponding to each of the synchronous `concurrency:copy` functions. Asynchronous copy APIs have similar copying semantics as their synchronous counterparts, with the exception that the return type is a `concurrency::completion_future` object that can be waited on, instead of `void` in the case of synchronous copy functions.

This example splits the input data over three iterations to perform color conversion on the input data. In each iteration of synchronous version (in the loop):

- The input data is first copied to the destination array using `concurrency::copy()`.
- The color conversion routine is run on the input, and output is generated.
- The resulting output is copied back to the host using `concurrency::copy()`.

In the asynchronous version (unrolled in code, no loop):

- Three asynchronous copies are scheduled using `concurrency:copy_aync` to copy input data to their respective destination arrays and completion_future returns are tracked.
- A `wait()` is called on the `completion_future` object before running the color conversion routine to ensure that the input data needed for the execution of that specific kernel call is available.
- An asynchronous copy is scheduled using `concurrency:copy_aync` to copy output data to its destination, and `completion_future` return is tracked.
- A `wait()` is called on each of the output `completion_future` objects to ensure the output data is in place.

In this example, `asyncArrayCopy()` is unrolled to make the code readable, thus limiting the input data split to three. However, it is possible to remove the limit on the split by implementing a ping-pong buffer concept for the input and output AMP arrays. This example shows that C++ AMP asynchronous data copies can be used efficiently to gain a noticeable performance boost over synchronous data copies. A simple color-conversion routine of RGB to YUV 4:4:4 is used to bring out the uses and advantages of asynchronous copies.

4 Recommended Input Option Settings

For best performance, enter the following on the command line: `-x 23592960 -q -t -e`

5 References

1. http://blogs.msdn.com/b/nativeconcurrency/archive/2012/07/17/choosing-between-array-and-array-view-in-c-amp.aspx
2. http://blogs.msdn.com/b/nativeconcurrency/archive/2012/01/10/transferring-data-between-accelerator-and-host-memory.aspx
3. http://en.wikipedia.org/wiki/YUV

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/
Forum: developer.amd.com/openclforum