AMD Accelerated Parallel Processing TECHNOLOGY

SAMPLE

Binary Search Device Side Enqueue

1 Overview

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. `BinarySearchDeviceSideEnqueue`
   This searches an element in an array of 64 elements.
2. `BinarySearchDeviceSideEnqueue -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 cpu or gpu.
-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 kernel.
-p	--platformId	Select the platformId to be used (0 to N-1, where N is the number of available platforms).
-d	--deviceId	Select deviceId to be used (0 to N-1, where N is the number of available devices).
-v	--version	AMD APP SDK version string.
-x	--length	Length of the input array.
-f	--find	Element to be found.
-s	--subdivisions	Number of subdivisions.

Short Form	Long Form	Description
`-i`	`--iterations`	Number of iterations for kernel execution.
`-k`	`--nKeys`	Number of Input Keys
`-eq`	`--devEnqueue`	Device-side kernel enqueue. To run the sample using OpenCL 1.x, choose `-eq 0`.

2 Introduction

It finds the position of a given element in a sorted array. If the element is not present in the array that is reported too. Instead of a binary search where the search space is halved every pass, we divide it into N segments and call it N'ary search. While plain binary search has a computation complexity of log to base 2, N'ary search is log to base N.

This sample demonstrates the device-side enqueue feature of OpenCL 2.0. The sample has 2 versions of Binary Search implementation. The first version is an OpenCL 1.2 implementation where the iterative algorithm is implemented by launching kernel in multiple passes. The second version uses the device-side enqueue feature of OpenCL 2.0 where the same iterative algorithm is implemented by enqueuing kernels on the device.

The device-side enqueue feature of OpenCL 2.0 allows a kernel to independently enqueue to the same device, without host interaction. A kernel may enqueue code represented by Block syntax, and control execution order with event dependencies including user events and markers.

3 Implementation Details

This is an N'ary search algorithm. For this particular implementation, the size of the array must be a multiple of 256. Consider 10000 (10^5) elements in sorted order from which an element must be searched. First, we divide the array into 10 segments of 10000 (10^4) elements; then, we find the segment to which the element belongs and further divide the segment into 10 segments of 1000 (10^3) elements. Thus, we narrow our search space by subdividing the array.

For example, assume your input array is 2, 4, ...$2*10^5$, and you are searching for 42:

The first pass consists of:

Thread 0: $2..2*10^4$	lower, upper bounds: 0, 10^4
Thread 1: $2*10^4+2..3*10^4$	lower, upper bounds: 10^4, $2*10^4$
Thread 2: $3*10^4+2..4*10^4$	lower, upper bounds: $2*10^4$, $3*10^4$
etc.	

The value 42 is not between the lower-bound and upper-bound of any thread other than thread 0. Thus, only thread 0 writes to the output buffer. It writes its own lower bound, upper bound, and, since 42 is not equal to the lower bound element (2), it writes 0 in the third element.

The output array is 0, 10^4, 0.

Similarly, the next pass has an output of 0, 10^3, 0. The pass after that has an output of 0, 10^2, 0.

Now the segment being searched in is 2, ... 200. Each segment is now 10 elements, so the threads are:

Thread 0: 2..20

Thread 1: 22..40

Thread 3: 42..60

This time only thread 3 writes to the output, and the third element is 1, meaning that the element is found.

The search is done, finding the index at which this element is present, and no further kernel calls are made.

If instead of 42 we were searching for 43, the subdivisions would go one step further, and the next pass would have 10 threads each being over a single element 42, 44, 46, etc.

Since 43 is not equal to any of them, and since the next subdivision's size is smaller than 1, the element can be said to not be present in the input array. So, no further kernel calls are made; the element has not been found.

Two versions of kernel are implemented. The first version is an iterative approach in which the host enqueues kernel in a loop after each subdivision. The second version uses the device-side enqueue feature of OpenCL2.0 to enqueue kernels on the device instead of returning control to the host.

Consider the same example stated above of searching number 42 in the range 0 to 10^5. The range is split into 10 segments . The number 42 falls on thread 0 in the range 0 to 10^4 . So a child kernel is launched in thread 0 with a new ndrange of 10^3. Further, another child kernel is launched in the range 0 to 10^2 . This process repeats until the search space becomes small. The final stage is carried out in the CPU to examine whether the given number is found or not, using search space of the last stage of device-side enqueue.

The BinarySearch_kernels.cl file contains two kernel implementations: `binarySearch_device_enqueue_Multikeys` and `binarySearch`. The sample implements the first kernel using the OpenCL 2.0 Device-Side Kernel enqueue feature. The second kernel is used in the sample to compare the performance between the OpenCL 2.0 and OpenCL 1.2 implementations.

4 References

1. http://en.wikipedia.org/wiki/Binary_search_algorithm
2. http://www.khronos.org/registry/cl/sdk/2.0/docs/man/xhtml/enqueue_kernel.html

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