AMD Accelerated Parallel Processing TECHNOLOGY

SAMPLE

DeviceEnqueueBFS

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.

Ensure that the OpenCL 2.0 environment is installed.

Type the following command(s).

1. `DeviceEnqueueBFS`
   This commands runs the program with the default options.
2. `DeviceEnqueueBFS -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 [cpu|gpu]`	Devices on which the OpenCL kernel 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.
`-v`	`--version`	AMD APP SDK version string.
	`--dump [filename]`	Dump the binary image for all devices.
	`--load [filename]`	Load the binary image and execute on the device.
	`--flags [filename]`	Specify the filename containing the compiler flags for building the kernel.
`-i`	`--iterations`	Number of iterations.
`-p`	`--platformId`	Select the platformId to be used[0 to N-1 where N is number platform s available].
`-d`	`--deviceId`	Select deviceId to be used[0 to N-1 where N is number devices available].

Short Form	Long Form	Description
`-n`	`--Matrix Size`	Dimension of an Adjacency Matrix
`-l`	`--localsize`	Number of work-items per work-group (should be 2 ^ N)

2 Introduction

This sample demonstrates the device-enqueue feature of OpenCL 2.0. This feature allows one to launch kernels from other kernels without host intervention. In OpenCL 1.2, launching kernels required host intervention.

The sample implements BFS, the most widely used graph searching algorithm to demonstrate this feature.

3 Implementation

This sample implements the Breath First Search (BFS) problem:

3.1 BFS Problem:

BFS is a strategy for searching in a graph. It begins at root node and inspects all the neighboring nodes. Then for each of those neighboring nodes in turn, it inspects their neighbor nodes which are unvisited and so on. It has the extremely useful property that if all the edges in a graph are un-weighted (or the same weight) then the first time a node is visited is the shortest path to that node from the source node.

3.2 Implementation

Considering BFS property, the sample uses a graph of same weight and gives the shortest path for each node from source node.

The sample uses compressed sparse row (CSR) sparse matrix format to represent the graph. This representation includes two arrays, *col_index*: stores list of neighbor nodes and *row_offset*: it stores the offset of the neighbor list for each node.

The sample implements parallel BFS using level synchronous operations on the GPU.

The device code processes all the vertices at a particular level in a parallel. It calls ***enqueue_kernel(),*** an OpenCL C function, to enqueue the block (list of neighbor nodes) for execution to device queue. In place of queue, it uses pipe memory objects to assemble all the neighbor nodes to be visited in next iteration.

It computes the distance of each node from the root node to verify CPU and GPU results.

3.2.1 Host Side

The sample first creates a command-queue on the device by using the `clCreateCommandQueueWithProperties()` function, by setting the `CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE | CL_QUEUE_ON_DEVICE` flag. The `clCreateCommandQueueWithProperties()` function also sets the

`CL_QUEUE_ON_DEVICE_DEFAULT` flag to make this queue the default device queue.

3.2.2 Kernel Side

The code includes two kernels.

1. The first kernel is the `writePipe` kernel. This kernel is used to initialize the pipe memory object with root node. It is called only once at very beginning.
2. The second kernel is the `deviceEnqueueBFS` kernel which does actual parallel BFS computation. It starts with root node. At each iteration, this kernel first reads vertices of the current level from read-pipe memory object and check whether these vertices are visited. If not, then kernel writes all its neighbor vertices into a write pipe memory object. After assembling the neighbor vertices into a write-pipe memory object, it launches the same kernel by swapping write pipe into read pipe. Kernel repeats the same process, till it visits all the vertices.

The sample also uses `atomic_fetch_add`, an atomic add operation, to compute the launch size for future launces of the kernel.

4 References

1.https://www.khronos.org/registry/cl/sdk/2.0/docs/man/xhtml/clCreateCommandQueueWithProperties.html

3. https://www.khronos.org/registry/cl/sdk/2.0/docs/man/xhtml/enqueue_kernel.html
4. http://www.systap.com/pubs/xdata/Large-Scale-Graph-Processing-Algorithms-on-the-GPU.pdf
5. http://en.wikipedia.org/wiki/Breadth-first_search

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