AMD **Accelerated**
Parallel Processing
TECHNOLOGY

SAMPLE

SVMAtomicsBinaryTreeInsert

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 and that the SVM Fine Grain Atomics feature is supported. Currently on AMD platforms, SVM Fine Grain Atomics feature is supported on only Linux 64-bit.

Type the following command(s).

1. `SVMAtomicsBinaryTreeInsert`
   This command runs the program with the default options.
2. `SVMAtomicsBinaryTreeInsert -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].

Short Form	Long Form	Description
`-d`	`--deviceId`	Select deviceId to be used[0 to N-1 where N is number devices available].
`-s`	`--seed`	Seed to the random number generator.
`-n`	`--numins`	Number of nodes to insert (< 125000000)
`-in`	`--initnodes`	Number of initial nodes (> 1)
`-hp`	`--host`	Percentage of nodes to be inserted on the host.
`-po`	`--prorder`	Print order of the tree nodes.

2 Introduction

Building a binary tree with a large number of nodes can be a very time consuming operation when executed on the CPU. Using OpenCL on GPUs, one can accelerate this task significantly. However implementing node insertion into a binary tree poses a few challenges with OpenCL 1.x.

1. The typical data structure of a binary tree includes pointers that point reference to left and right child nodes. Passing data structures with pointers is not allowed in OpenCL 1.x platforms. Hence, the only way to work around this in OpenCL 1.x is to transform the pointer based data structure to indices based structure and pass it to OpenCL kernels. Once a kernel operation is complete, the data must be transformed back to pointer based structure to be used on the host. These data transform operations can be very costly, and affect the overall performance. The code complexity also increases due to the additional data transformations.
2. Furthering accelerate the node insertion operation by having the host and OpenCL device simultaneously inserting nodes into the tree is not possible with OpenCL 1.x. This is because, for concurrent node insertion, synchronization between the host and the OpenCL device is required and there is no way to do this in OpenCL 1.x.

OpenCL 2.0 helps overcome both the above limitations. It supports passing pointer-based data structures to kernels and also enables synchronization between the host and the OpenCL device through C++11 Atomics. It helps in accelerating node insertion and at the same time improves programmability.

3 Implementation

The sample workflow is as follows:

1. Host creates binary tree with a given number of initial nodes.
2. The tree structure is created as a SVM Fine Grain buffer with Atomics flag. This is required for the atomic operations to work.
   ```
   int flags = CL_MEM_READ_WRITE | CL_MEM_SVM_FINE_GRAIN_BUFFER |
   CL_MEM_SVM_ATOMICS;
   node* svmTreeBuf = (node *) clSVMAlloc(context,   flags,
   total_nodes*sizeof(node),   0);
   ```
3. The sample reads the total number of nodes ,’ n’, to be further inserted into the tree.
4. The ‘n’ nodes to be inserted can be done all on OpenCL device or all on host or a combination of both. This is decided based on the value of host compute percentage (-hp) passed as input. For example, if –hp value is 0, entire node insertion happens on OpenCL

device while if its 100 its completely on host. If –hp is say, 10, host inserts 10% of the total nodes to be inserted and device the remaining 90%.

5. The host code uses OpenMP to parallelize the host part of insertion operation.
6. C++ Atomic operations are used on both the host and the OpenCL kernel to ensure that at any given instance of time, only one thread (either on the host or on the device) inserts to a given parent node.

4 References

1. Memory Model, section 3.3 and Share Virtual Memory, section 5.6 in OpenCL 2.0 specficiation
2. Atomic functions, section 6.13.11, in OpenCL 2.0 C specifiction
3. Atomic Weapons: The C++ Memory Model and Modern Hardware. Herb Sutter. http://herbsutter.com/2013/02/11/atomic-weapons-the-c-memory-model-and-modern-hardware