AMD **Accelerated** Parallel Processing TECHNOLOGY

SAMPLE

Structure of Array versus Array of Structure

1 Overview

1.1 Location

`$<APPSDKSamplesInstallPath>\samples\opencl\cpp_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. `SoAversusAoS`
   This runs the program with the default options `-s 4096 -n 4096`.
2. `SoAversusAoS -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

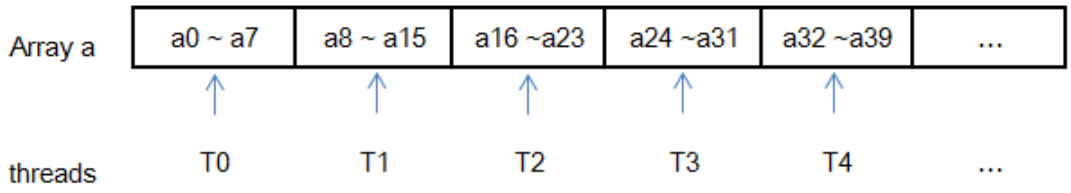
Short Form	Long Form	Description
`-h`	`--help`	Shows all command options and their respective meaning.
	`--dump`	Dump binary image for all devices.
	`--load`	Load binary image and execute on device.
	`--flags`	Specify compiler flags to build kernel.
`-p`	`--platformId`	Select the platformId to be used (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.
`-i`	`--iterations`	Number of iterations for kernel execution.
`-s`	`--size`	Tree size.
`-n`	`--number`	Number of trees.

2 Introduction

This sample give us a comparison between AOS (array of structure) and SOA (structure of array). It presents that the mode of accessing global memory can affect the memory access time.

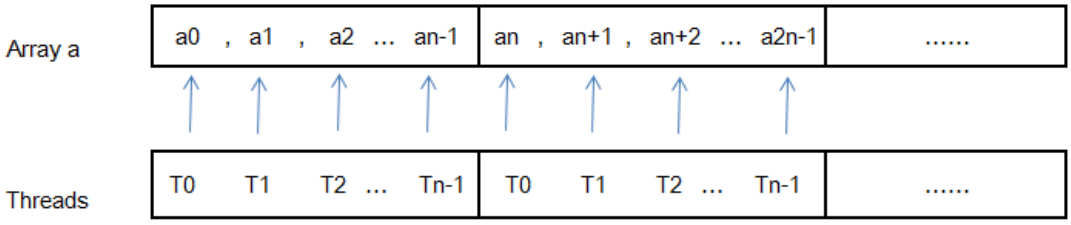
2.1 Array of Structure (AOS)

For the AOS mode to access memory, one thread accesses the adjacent memory. So, all the threads access non-adjacent memory at the same time. If one thread accesses eight numbers of an array, the AOS mode looks like this:



2.2 Structure of Array (SOA)

For SOA mode, adjacent threads access adjacent memory at the same time, and the AOS looks like this:



3 Implementation Details

To intuitively understand SOA and AOS from the code, consider workers picking apples form the trees.

Imagine an apple tree that has multiple branches with apples. This could be represented by a single structure (tree) with multiple data members (branches with apples). There are many trees in an orchard (array), so that can be an array of structures. If one worker/thread is assigned to work on one tree, that is like accessing adjacent memory locations sequentially. This access pattern is inefficient on the GPU because there is a large memory stride between adjacent threads.

```
template <int TREE_SIZE>
struct AppleTree
{
    int apples[TREE_SIZE];
};
```

It is more efficiently to have adjacent threads to access adjacent memory locations. This pattern is known as coalesced access. To achieve that, the data is reorganized in an SOA way.

Using the apple tree example, a structure now represents one particular branch for all the trees in the orchard (array). Each tree has multiple branches, so an array of this kind of structure is used to represent the entire orchard (AOS). If the data is organized in this way, adjacent workers/threads w access adjacent memory locations, thus getting higher performance due to coalesced accesses.

To define a data type in the kernel for SOA:

```
template <int TREE_NUM>
struct ApplesOnTrees
{
    int trees[TREE_NUM];
};
```

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