ATI STREAM COMPUTING SAMPLE

Binary Search

1 Overview

1.1 Location

`$(ATISTREAMSDKSAMPLESROOT)\samples\opencl\cl\app`

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 `$(ATISTREAMSDKSAMPLESROOT)\samples\opencl\bin\x86\` for 32-bit builds, and `$(ATISTREAMSDKSAMPLESROOT)\samples\opencl\bin\x86_64\` for 64-bit builds.

Type the following command(s).

1. `BinarySearch`
   This searches an element in an array of 64 elements.
2. `BinarySearch -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.
-q	--quiet	Quiet mode. Suppresses all text output.
-e	--verify	Verify results against reference implementation.
-v	--verbose	Verbose output.
-t	--timing	Print timing.
-x	--width	Width of problem domain.
-y	--height	Height of problem domain.
-z	--depth	Depth of problem domain.
-f	--find	Element to be found.
	--device	Devices on which the program is to be run. Acceptable values are cpu or gpu.

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.

3 Implementation Details

This is an N'ary search algorithm. Consider 10000 (10^5) elements in sorted order in 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, we can now say that the element is not present in the input array. So, no further kernel calls are made; the element has not been found.

Contact

Advanced Micro Devices, Inc.
One AMD Place
P.O. Box 3453
Sunnyvale, CA, 94088-3453
Phone: +1.408.749.4000

For Stream Computing:
URL: www.amd.com/stream
Questions: streamcomputing@amd.com
Developing: ATI_Stream_SDK_Help_Request
Forum: www.amd.com/streamdevforum