AMD **Accelerated** Parallel Processing TECHNOLOGY

SAMPLE

K-means Auto-clustering

1 Overview

1.1 Location

`$<AMDAPPSDKSamplesInstallPath>\samples\opencl\cl\1.x`

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
`$<AMDAPPSDKSamplesInstallPath>\samples\opencl\bin\x86\` for 32-bit builds, and
`$<AMDAPPSDKSamplesInstallPath>\samples\opencl\bin\x86_64\` for 64-bit builds.

Type the following command(s).

1. `KmeansAutoclustering`
   This command runs the program with the default options.
2. `KmeansAutoclustering -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`	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-related statistics.
	`--dump`	Dump binary image for all devices.
	`--load`	Load binary image and execute on device.
	`--flags`	Specify the compiler flags for building the kernel.
`-x`	`--points`	Number of points in the given 2D plane.
`-k`	`--clusters`	Number of clusters into which to cluster the inputs points, using the K-means algorithm. If k is explicitly specified, auto-clustering is disabled.
`-lk`	`--lboundnumclusters`	Lower bound to start testing for auto-clustering. Valid only if the -k option is not used.
`-uk`	`--uboundnumclusters`	Upper bound to stop testing for auto-clustering. Valid only if the -k option is not used.

Short Form	Long Form	Description
`-ck`	`--numcustomclusters`	This option customizes the creation of input points. -ck is used to specify the number of clusters to be created for input at random places in the 2D plane. If -ck is 0, random data is created.
`-p`	`--platformId`	Select platformId to be used (0 to N-1, where N is the number of available platforms).
`-v`	`--version`	AMD APP SDK version string.
`-d`	`--deviceId`	Select deviceId to be used (0 to N-1, where N is the number of available devices).
`-i`	`--iterations`	Number of iterations.

2 Introduction

This sample aims at clustering a given set of 2D points into the most fitting number of clusters using the K-means clustering algorithm and the Silhouette method.

In this implementation, a set of 2D points is clustered by using OpenCL kernels and the clusters are displayed in real-time (as iterative refinement happens) using OpenGL.

2.1 K-Means Algorithm

K-means, a Cluster- analysis method frequently used in data-mining, partitions a set of N points into K clusters by grouping the points based on closeness. The method finds applications in areas such as vector quantization, density estimation, workload behavior characterization, and image compression.

Every cluster is characterized by a parameter called the "Centroid." The centroid can be looked at as the central defining point of a cluster. For a given cluster containing "n" 2D points, the centroid is defined as follows:

$$(C_x, C_y) = \left(\frac{x_1 + x_2 + x_3 + ..x_n}{n}, \frac{y_1 + y_2 + y_3 + ..y_n}{n}\right)$$

The most popular algorithm used to compute K-Means is the Lloyd's algorithm. The Lloyd’s algorithm is based on iterative-refinement and is listed below:

1. Select K random points as initial centroids.
2. Repeat until there is no further change in the centroid position of the clusters:
    i. Assign each point to its closest centroid to form K clusters.
    ii. Recompute the centroid of each cluster.

2.2 Auto-clustering using the Silhouette technique

Silhouette is a method for the interpretation and validation of clustering.

Let us assume that the data have been clustered via any technique, such as k-means. For each datum `i`, let `a(i)` be the average dissimilarity of `i` with all the other data points within the same cluster. We can interpret `a(i)` as how well matched `i` is to the cluster it is assigned (the smaller the value, the better the matching). Then find the average dissimilarity of `i` with the data of another single cluster. Repeat this for every cluster, of which `i` is not a member. Denote the lowest average dissimilarity to `i` of any such cluster by `b(i)`. The cluster with this lowest average dissimilarity is said to be the "neighboring cluster" of `i`. We now define the Silhouette score for `i`, `s(i)`, as follows:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

From the above definition it is clear that:

$$-1 \leq s(i) \leq 1$$

For `s(i)` to be close to 1 we require:

$$a(i) << b(i)$$

As `a(i)` is a measure of how dissimilar `i` is to its own cluster, a small value means it is well matched. Furthermore, a large `b(i)` implies that `i` is badly matched to its neighboring cluster. Thus, if `s(i) is` close to 1, the datum is appropriately clustered. If `s(i)` is close to -1, then by the same logic we see that `i` could be more appropriately clustered in its neighboring cluster. A `s(i)` value of near zero means that the datum is on the border of two natural clusters.

3 Implementation

The sample implements two algorithms:

1. K-means algorithm (Lloyd's algorithm)
2. Silhouette algorithm

3.1 K-means algorithm implementation

Let "N" denote the number of points in the 2D plane and "K" the number of clusters to be formed. As mentioned in the algorithm description above, the centroids are calculated iteratively until there is no further change in their position. In every iteration, the centroid positions change depending on how many points fall into that particular cluster. Each iteration is divided into two steps:

1. For every point in the plane, calculate the distance from all the centroids and choose the closest centroid.

2. For every centroid, find the points that are close to that centroid. Take an average of all these points. This will be the new centroid position.

The first step requires N global work-items each working on a point. Every point will compare its distances from all the centroids and will choose the closest centroid. Two temporary buffers "bin" and "counter" of size K (for every centroid) are used. Every time a point chooses a centroid, say `c`, `bin[c]` is accumulated with the distance of that point with the chosen centroids. This operation has to be atomic. `Counter[c]` is also incremented, which is also atomic. Since the coordinates are floating point and OpenCL does not yet support atomic accumulations on floating points, the atomic accumulations are implemented by using `atomic_cmpxchgs()`.

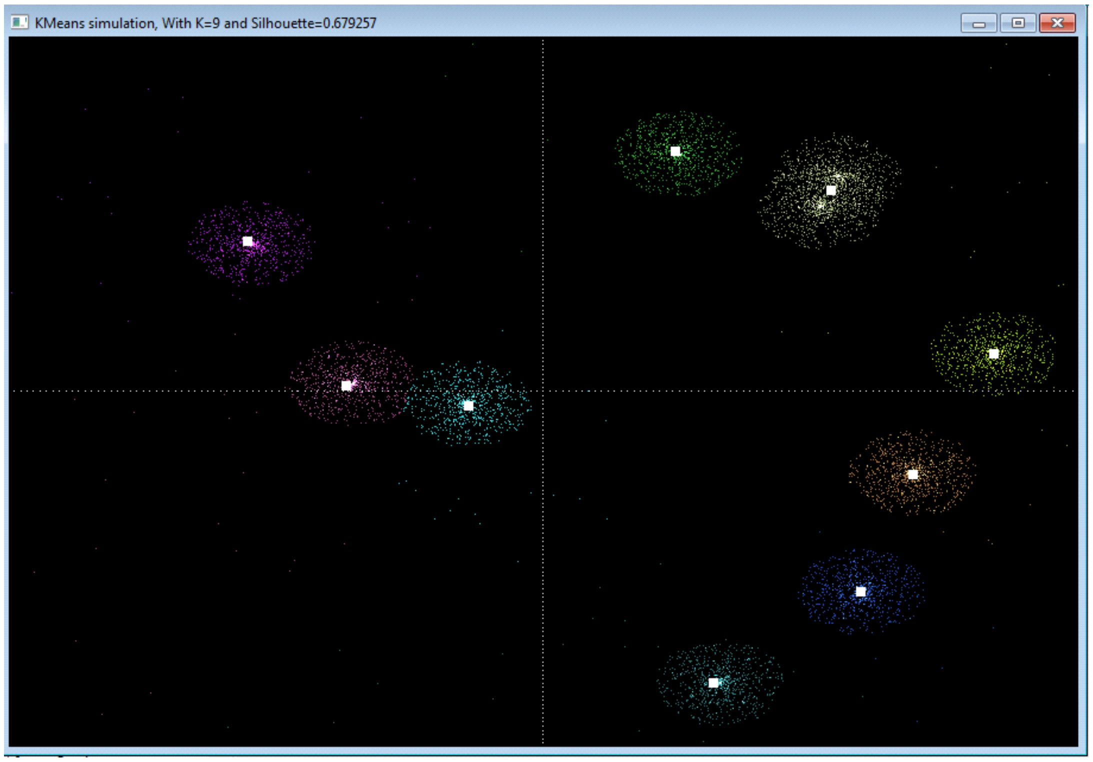
The second step requires `K` iterations working on each cluster. For every cluster `c`, `(bin[c]/counter[c])` is calculated to find the new centroids. These centroids are then fed into the first step. The process goes on iteratively until the centroids become stable.

Note: This second part is implemented in the CPU as the value of `K` is generally very small. In the sample, the max value of `K` that is supported is 16.

3.2 Silhouette algorithm implementation

Here it is assumed that the input has already been clustered, for a fixed value of `K`. `N` threads are spawned, and each thread is responsible for computing the silhouette score of a single point. The threads also need to add their silhouette score to a global variable, which is later averaged out to compute the Silhouette for the complete clustering.

In both the algorithms, LDS buffers are used to ease the pressure of global atomic writes. Workgroups write into their own LDS space initially using local atomics. Then a single work-item from the workgroup writes this value in the global buffer using global atomics.



4 References

1. http://en.wikipedia.org/wiki/K-means_clustering.
2. http://en.wikipedia.org/wiki/Silhouette_(clustering).
3. http://www.sciencedirect.com/science/article/pii/0377042787901257.

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**

The contents of this document are provided in connection with Advanced Micro Devices, Inc. ("AMD") products. AMD makes no representations or warranties with respect to the accuracy or completeness of the contents of this publication and reserves the right to make changes to specifications and product descriptions at any time without notice. The information contained herein may be of a preliminary or advance nature and is subject to change without notice. No license, whether express, implied, arising by estoppel or otherwise, to any intellectual property rights is granted by this publication. Except as set forth in AMD's Standard Terms and Conditions of Sale, AMD assumes no liability whatsoever, and disclaims any express or implied warranty, relating to its products including, but not limited to, the implied warranty of merchantability, fitness for a particular purpose, or infringement of any intellectual property right.

AMD's products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD's product could create a situation where personal injury, death, or severe property or environmental damage may occur. AMD reserves the right to discontinue or make changes to its products at any time without notice.

Copyright and Trademarks
© 2015 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, ATI, the ATI logo, Radeon, FireStream, and combinations thereof are trademarks of Advanced Micro Devices, Inc. OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos. Other names are for informational purposes only and may be trademarks of their respective owners.