AMD Accelerated Parallel Processing TECHNOLOGY

SAMPLE

Floyd-Warshall

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.

Type the following command(s).

1. `FloydWarshall`
   Calculates the shortest distance between each pair of 64 nodes of a graph whose weights are initialized randomly.
2. `FloydWarshall -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

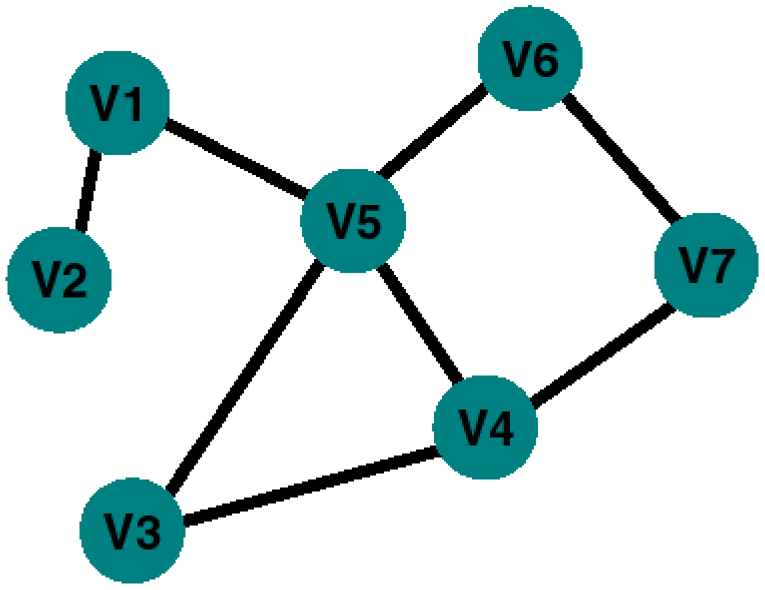
Short Form	Long Form	Description
`-h`	`--help`	Shows all command options and their respective meaning.
	`--device`	Devices on which the program is to be run. Acceptable values are `cpu` or `gpu`.
`-q`	`--quiet`	Quiet mode. Suppresses all text output.
`-e`	`--verify`	Verify results against reference implementation.
`-t`	`--timing`	Print timing.
	`--dump`	Dump binary image for all devices.
	`--load`	Load binary image and execute on device.
	`--flags`	Specify compiler flags to build the kernel.
`-p`	`--platformId`	Select 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.
`-x`	`--nodes`	Number of nodes.
`-i`	`--iterations`	Number of iterations for kernel execution.

2 Introduction

A graph is an essential tool to model and understand both natural and man-made phenomena. Graphs are frequently encountered when solving problems such as Internet routing, supply-chain networks, oil pipelining, electrical grids, VLSI fabrication, and social networks. The unifying motivation for all such problems is the minimization of energy spent in moving material or information from source to destination through an intervening network of entities. A commonly occurring problem in graph theory is to find the *shortest path* between two entities. Before elaborating the concept of a shortest path, we recall the formal definition of a graph as understood in mathematics and computer science:

> A *graph* is a collection of vertices and edges that maybe denoted as *G(V,E)*, where V = {V_1, V_2, …, V_n} is a set of vertices or nodes, and E = {e_1, e_2, …, e_m} is a set of edges, where each edge is an unordered pair (ordered pair for directed graphs) of vertices. An edge e_i connecting the vertices V_p and V_q is denoted by (V_p, V_q).

Figure 1 shows a diagram of a graph with seven vertices and eight edges.



G(V,E)

V = { V_1, V_2, V_3, V_4, V_5, V_6, V_7 }

E = { (V_1,V_2), (V_1,V_5), (V_5,V_3), (V_5,V_4), (V_5,V_6), (V_6,V_7), (V_7,V_4), (V_4,V_3) }

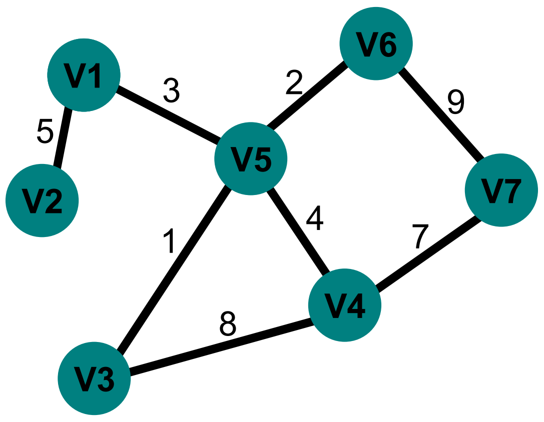
Figure 1 Graph G(V,E): V has Seven Vertices, and E has Eight Edges

3 Shortest Path

A *weighted graph* is a graph where every edge has a weight attached to it. The weight generally signifies some “cost” or “distance” associated with that edge. The *shortest path problem* tries to find the path between a given pair of vertices that minimizes the sum of the weights of edges encountered along that path. Shortest path problems can be formulated for a single source or a single destination. A single-source shortest path problem finds the shortest path from a given source vertex to all other vertices; a single-destination problem finds shortest paths from all vertices to a specified one. Alternatively, we can find shortest paths from every vertex in a graph to every other vertex. This is termed as the *all pair shortest path* problem and is the premise of the Floyd-Warshall algorithm.

4 Adjacency Matrix

The *adjacency matrix* of a graph is a square matrix of dimensions *n x n*, where *n* is the number of nodes in the graph. If P(nxn) is the adjacency matrix for graph G(V,E), then P(i,j) indicates the weight of the edge from V_i to V_j. Figure 2 shows the graph in Figure 1 with each edge labeled by a weight and the corresponding adjacency matrix.



$$\begin{array}{c|ccccccc} & V_1 & V_2 & V_3 & V_4 & V_5 & V_6 & V_7 \\ \hline V_1 & 0 & 5 & \infty & \infty & 3 & \infty & \infty \\ V_2 & 5 & 0 & \infty & \infty & \infty & \infty & \infty \\ V_3 & \infty & \infty & 0 & 8 & 1 & \infty & \infty \\ V_4 & \infty & \infty & 8 & 0 & 4 & \infty & 7 \\ V_5 & 3 & \infty & 1 & 4 & 0 & 2 & \infty \\ V_6 & \infty & \infty & \infty & \infty & 2 & 0 & 9 \\ V_7 & \infty & \infty & \infty & 7 & \infty & 9 & 0 \end{array}$$

Figure 2 Graph G(V,E) with Weights Attached to Each Edge (Left) and the Corresponding Adjacency Matrix (Right)

Note that the weight between a pair of nodes is infinity (∞) if there is no edge between them. Also note that zeroes appear along the diagonal. The weight of an edge from a node to itself is zero because there is no cost in the traversal.

5 Floyd-Warshall Algorithm

The Floyd-Warshall algorithm computes the shortest path between each pair of nodes in a graph (see reference [1]). It is a dynamic programming approach that iteratively refines the adjacency matrix of the graph in question until each entry in the matrix reflects the shortest path between the corresponding nodes. The main idea of the algorithm is as follows: Given the shortest path between node V_i and V_j using $V_1 \ldots V_k$ as intermediate nodes, find out the shortest path between V_i and V_j using $V_1 \ldots V_{k+1}$ as intermediate nodes. This idea can be recursively formulated as:

ShortestPath(i, j, k) = min(ShortestPath(i, j, k-1) , ShortestPath(i, k, k-1) + ShortestPath(k, j, k-1))
ShortestPath(i, j, 0) = EdgeCost(i, j)

6 Implementation Details

In the OpenCL implementation of the Floyd-Warshall algorithm, we start with a randomly generated adjacency matrix. It is a full matrix in the sense that there is an edge from every node to every other node in the graph, which makes the graph a clique; also, it is a bidirectional graph. The weights of self-loops are set to zero, so zeroes appear along the diagonal of the matrix.

The inputs to the OpenCL kernel are the path distance matrix, path matrix and the step (iteration) number. The OpenCL kernel is invoked n times, where n is the number of nodes in the graph. At the k^{th} step, the kernel computes two numbers for every pair of nodes in the graph. The “direct distance” between them is determined by looking up the path distance matrix, and the “indirect distance” is computed by using node k as an intermediate node. If the shortest path between the nodes after k^{th} iteration passes through k, then the path matrix is updated with k. During the initial pass, k=1, the direct distance is simply the weight of the edge between V_i and V_j. The indirect distance using V_1 as an intermediate node is the sum of distances between V_i, V_1 and V_1, V_j. The smaller of the two distances is written back to the path distance matrix. If the shortest distance between V_i and V_j passes through V_1, the path matrix corresponding to the nodes V_i and V_j is updated with V_1. At the end of pass 1, the path distance matrix reflects the distances between each pair of nodes in the graph using V_1 as the intermediate node. This matrix is used as the basis for the next pass, where the same computation is done with respect to V_2. The final path distance matrix reflects the lengths of the shortest paths between each pair of nodes in the graph. The final path matrix reflects the intermediate nodes through which the shortest path between any two pairs of nodes passes.

7 References

1. http://en.wikipedia.org/wiki/Floyd-Warshall_algorithm

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**