National Tsing Hua University
Department of Electrical Engineering
11020EE521800 Application Acceleration with High-Level Synthesis LabC
Spring 2021
Vitis Data Analytics Library (Laboratory Proposal)
109061564 馬婕芸 110061551呂易縉 110061540賴聖耘

I. Introduction

Inside Vitis data analytics library, there are several APIs for accelerating data analytics applications, e.g. Data Mining APIs, Text processing APIs , DataFrame APIs, GeoSpatial APIs. We focus on data mining APIs since we need to use them to make classification, clustering, and regression. Like most other Vitis sub libraries, Data Analytics Library also organizes its APIs by levels which are shown below.

- The bottom level, L1, is mostly hardware modules with its software configuration generators.
- The second level, L2, provides kernels that are ready to be built into the FPGA binary and invoked with standard OpenCL calls.

II. Objectives

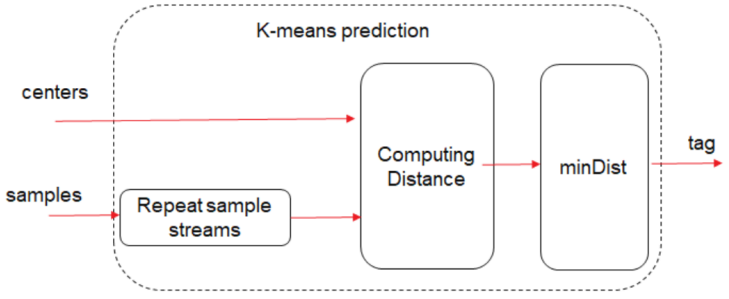
Among Data Analytics Library, we choose to implement the k-means clustering algorithm. K-means clustering is one of the oldest and most commonly used clustering algorithms, which is used to fund groups which have not been explicitly labeled in the data. It can be applied to many fields, including marketing, bioinformatics, computer vision, and etc. Vitis data analytics library organizes its APIs into 3 levels, including L1 for kernels HLS, L2 for integrators using C++ APIs. As a result, in this lab, we will implement the k-means clustering algorithm and verify it with the given dataset.

III. Background Introduction

Mathematically, the k-means clustering algorithm is shown below. Given a set of observations X, k-means clustering aims to partition the n observations into k sets S to minimize the variance of S.

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 = \arg\min_{\mathbf{S}} \sum_{i=1}^{k} |S_i| \operatorname{Var} S_i$$

Inside Vitis data analytics library, the algorithm is implemented in the following way. We need to provide an initial center setting of each cluster, and the predictor will iteratively update the distance between clusters to find the minimum distance as a solution.



In this algorithm, each iteration the Euclidean distance of each data point should be re-calculated and the calculation process is the same in each iteration. It is beneficial to use acceleration in this algorithm to make its application perform better.

IV. Implementation

K-means clustering algorithm implementation flow will be organized into 3 levels. L1 contains HLS-based unit tests for kernel primitives, and L2 contains the host-callable kernels to be built into FPGA binary and invoked with standard OpenCL calls. The implementation aims to change computational complexity O(Nsample * Kcluster * Dim * maxIter) to O(Nsample* (Kcluster/KU)*(Dim/DV)*maxIter) by accelerating calculating distances.

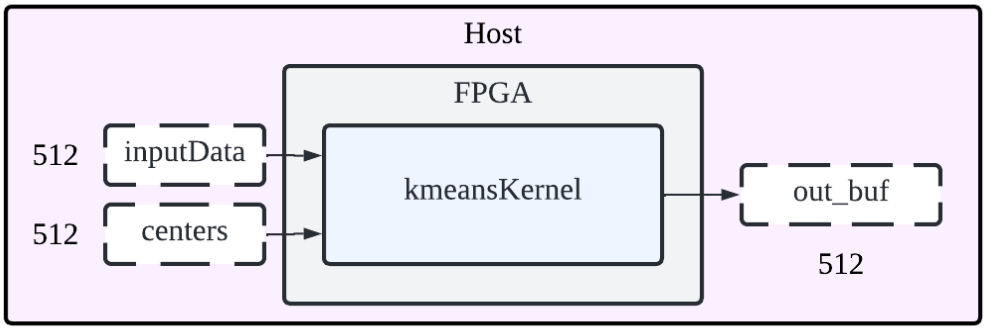
Dataset:

For verification, three open-source iris dataset will be used. There are 150 data samples in the dataset, which contains sepal length, sepal width, petal length, petal width and iris's class label. The dataset's structure is arranged as follows.

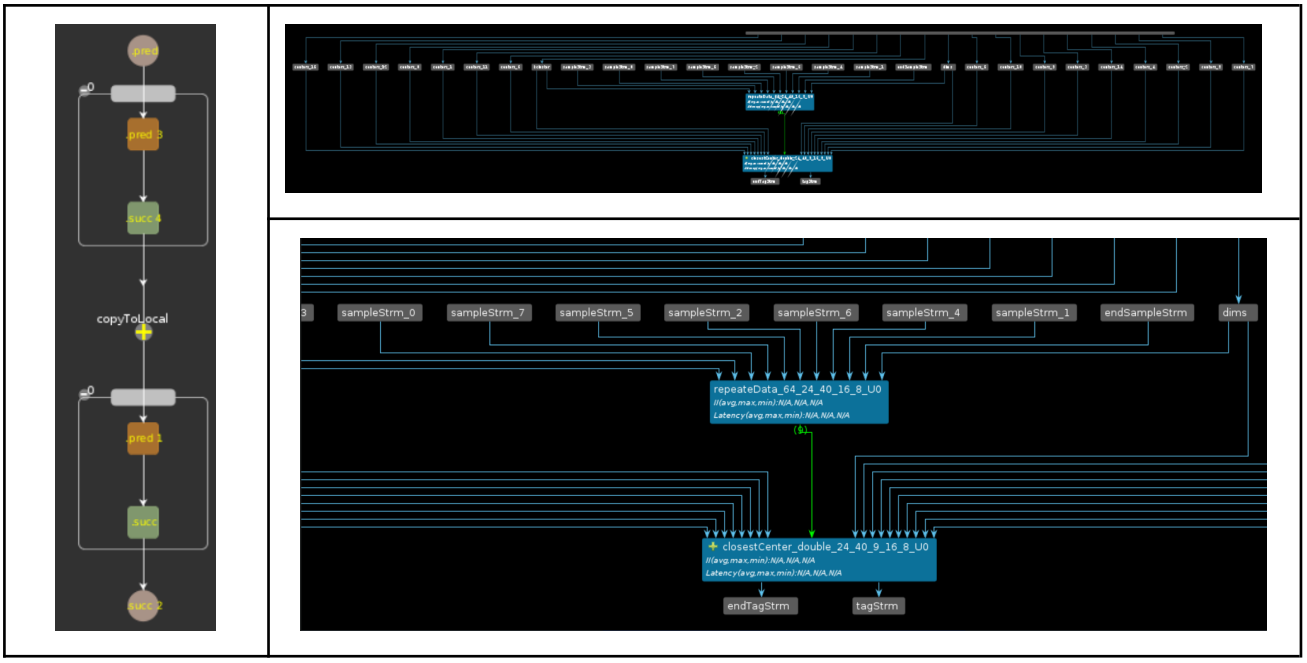
Sample #	Sepal length	Sepal width	Petal length	Petal width	Class label
0	5.1	3.5	1.4	0.2	0 (Setosa)
1	4.9	3.0	1.4	0.2	0 (Setosa)
2	4.7	3.2	1.3	0.2	0 (Setosa)
…	…	…	…	…	…
149	5.9	3.0	5.1	1.8	2 (Virginica)

Function hierarchy (dataflow and control flow):

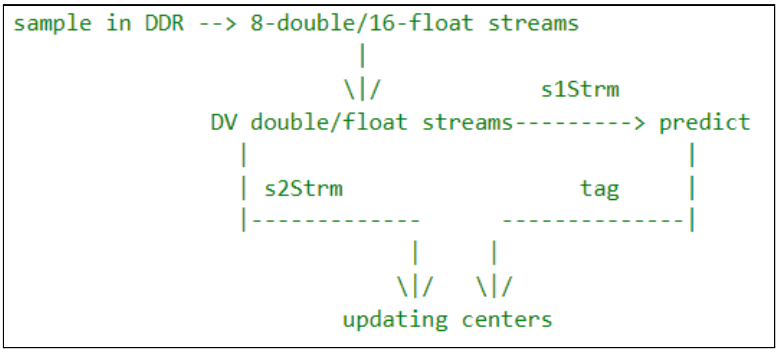
First of all, inside the L2 implementation, the block diagram inside the host program is shown below. The inputData is a preprocessed iris dataset, and the preprocessing is done by the host program. The centers are the predefined parameters. The output is the out_buf.



For the kmeansKernel, it contains three parts, including streaming the data from DDR to kmeansPredict, kmeansPredict kernel and output data stream from kmeansPredict to update center. The following figure shows the dataflow and control flow figures of kmeansPredict kernel which is implemented in L1. This kernel mainly calculates the distance of each sample and all centers.

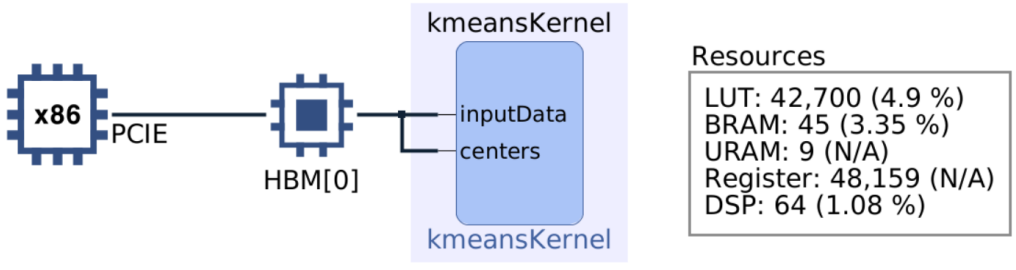


As for L2 implementation, the dataflow is shown below. The depth of s2 is bigger than s1 because kMeansPredict has a ping-pong buffer to repeat output each sample and each sample is used for updating centers until it is predicted.

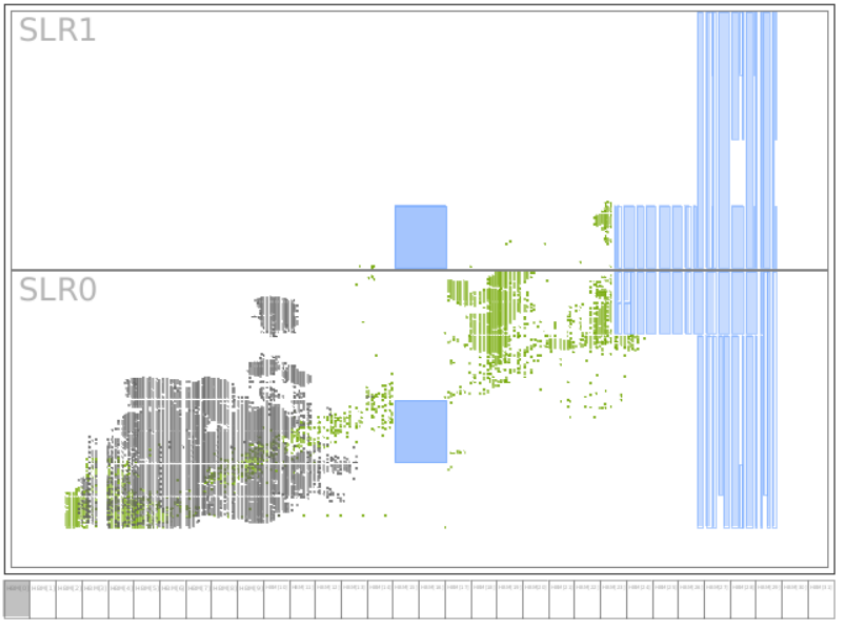


System block diagram:

When we implement the whole system on the U50 platform, the system diagram is shown below. The system takes up 42,700 LUTs, 45 BRAMs, 9 URAMs, 48,159 registers and 64 DSPs inside the FPGA. Besides, it also uses one HBM port to communicate with the x86 processor through PCIe.



From the utilization report, we can see that this FPGA is totally enough to implement this algorithm. Also, we can use this utilization proportion to calculate the resources we need when scaling the k-means kernel. After implementing the hardware target, we can see the real placing and routing results in the device map which is shown below.



Development Process:

When we implemented the whole system on the U50 platform, the only bug we met was the environment parameter called LC_ALL used in the workstation. After adding the `setenv LC_ALL "C"`, everything goes well. However, when running XRT simulation, since the workstation has limited memory, the process sometimes fails.

Lab Current Status:

L1: c-sim and co-sim are completed.
L2: c-sim, co-sim and hw FPGA are completed.
L3: this lab does not contain L3.

L2 kMeansTrain:

Kernel Templates in `xf::data_analytics::clustering`

kMeansTrain

```
#include "xf_data_analytics/clustering/kmeansTrain.hpp"
```

```
template <
    typename DT,
    int Dim,
    int Kcluster,
    int KU,
    int DV = 128 / KU
    >
void kMeansTrain (
    ap_uint <512>* data,
    ap_uint <512>* kcenters
    )
```

Input data includes:

1. Dynamic configuration in data[0], including the number of samples,the number of dimensions,the number of clusters,the maximum number of iterations,the distance threshold used for determining whether the iteration is converged.
2. Initial centers, which are provided by the host and compressed into many 512-bit packages.
3. Samples used for training,which are also compressed. kcenters is used for output best centers only.

Parameter description in KmeansTrain function:

Parameters:

DT	data type, supporting float and double
Dim	the maximum number of dimensions,dynamic number of dimension should be not greater than the maximum.
Kcluster	the maximum number of cluster,dynamic number of cluster should be not greater than the maximum.
KU	unroll factor of Kcluster, KU centers are took part in calculating distances concurrently with one sample. After Kcluster/KU+1 times at most, ouput the minimum distance of a sample and Kcluster centers.
DV	unroll factor of Dim, DV elements in a center are took part in calculating distances concurrently with one sample.
data	input data from host
kcenters	the output best centers

L1 kMeansPredict templates:

kMeansPredict predicts cluster index for each sample. In order to achieve acceleration, please make sure to partition 1-dim of centers.

Hardware Functions in `xf::data_analytics::clustering`

kMeansPredict

```
#include "xf_data_analytics/clustering/kmeansPredict.hpp"
```

```
template <
    typename DT,
    int Dim,
    int Kcluster,
    int uramDepth,
    int KU,
    int DV
    >
void kMeansPredict (
    hls::stream <ap_uint <sizeof (DT)*8>> sampleStrm [DV],
    hls::stream <bool>& endSampleStrm,
    ap_uint <sizeof (DT)*8*DV> centers [KU][uramDepth],
    const int dims,
    const int kcluster,
    hls::stream <ap_uint <32>>& tagStrm,
    hls::stream <bool>& endTagStrm
    )
```

Parameter description in KmeansPredict function:

Parameters:

DT	data type, supporting float and double
Dim	the maximum number of dimensions,dynamic number of dimension should be not greater than the maximum.
Kcluster	the maximum number of cluster,dynamic number of cluster should be not greater than the maximum.
uramDepth	the depth of uram where centers are stored. uramDepth should be not less than ceiling(Kcluster/KU) • ceil(Dim/DV)
KU	unroll factor of Kcluster, KU centers are took part in calculating distances concurrently with one sample. After Kcluster/KU+1 times at most, ouput the minimum distance of a sample and Kcluster centers.
DV	unroll factor of Dim, DV elements in a center are took part in calculating distances concurrently with one sample.
sampleStrm	input sample streams, a sample needs ceiling(dims/DV) times to read.
endSampleStrm	the end flag of sample stream.
centers	an array stored centers, user should partition dim=1 in its defination.
dims	the number of dimensions.
kcluster	the number of clusters.
tagStrm	tag stream, label a cluster ID for each sample.
endTagStrm	end flag of tag stream.

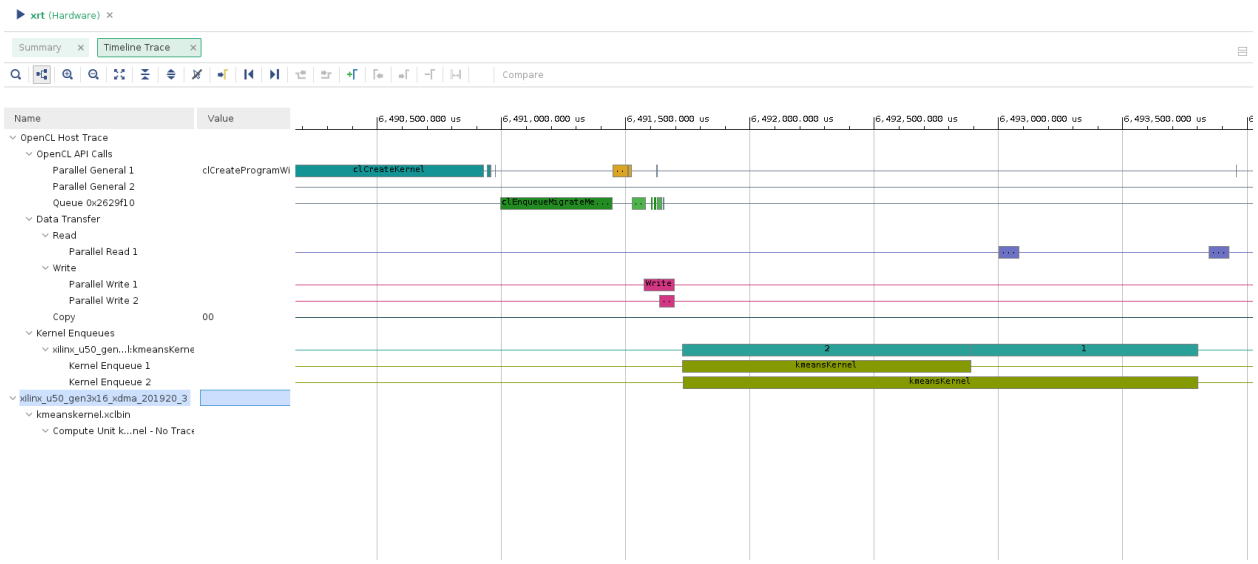
V. Analysis and Suggestion for Improvement

Resource Usage:

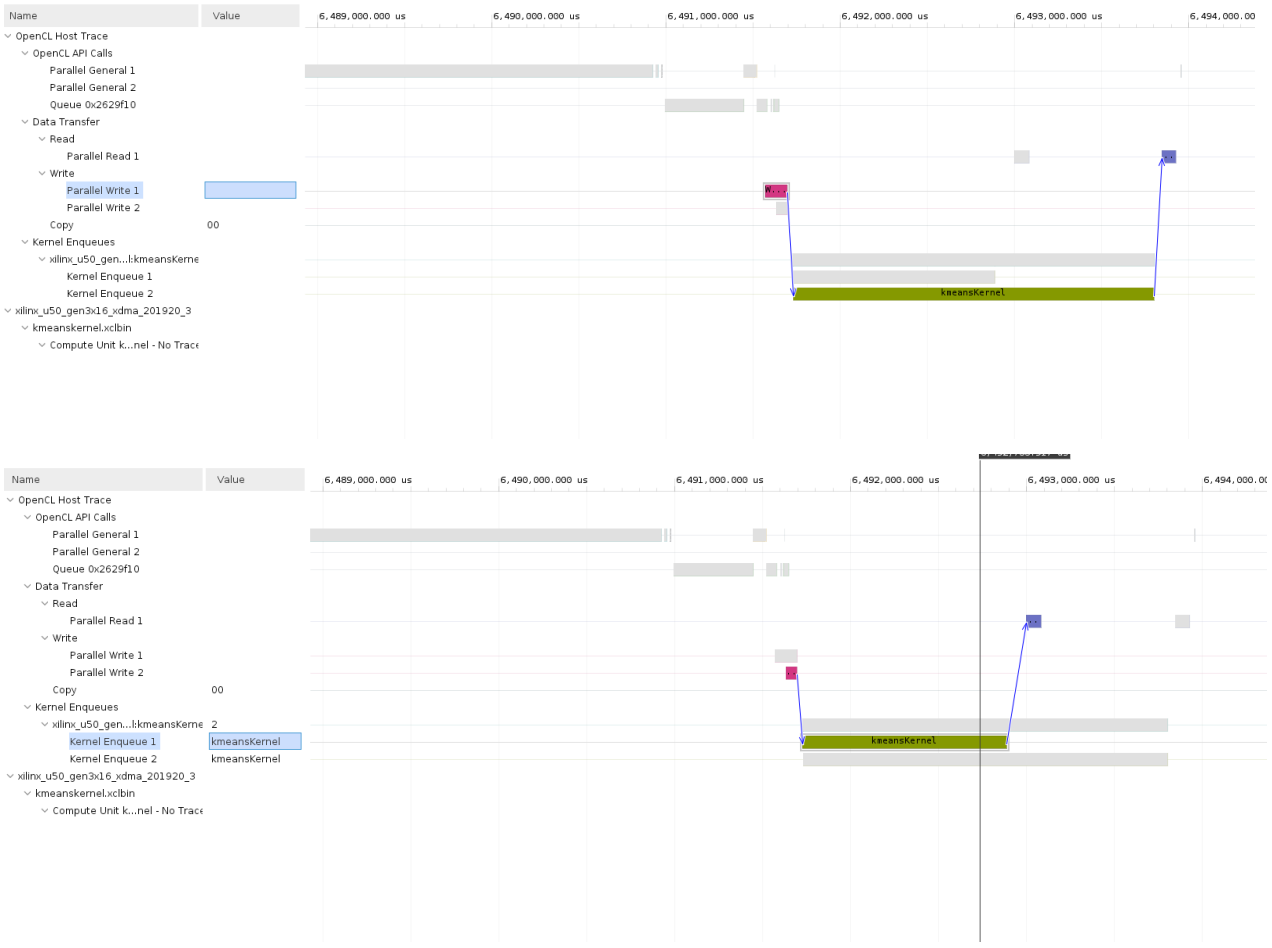
From the following figure, we can see that the iteration latency of AVEG_LOOP is very long and does not use any pipeline. This means that each interaction is executed sequentially. However, since the average value of each cluster can be calculated individually. We think this averaging process can also be implemented in the pipeline, though it may cause more resources. To implement this suggestion, we need to unroll the averaging loop and set the #pragma HLS pipeline.

Name	Issue Type	Latency (cycl...	Latency (ns)	Iteration Latency	Interval	Trip Cou...	Pipelined	BRAM	BRAM (%
kmeansKernel							no	60	
scanAndStoreCenters_double_16_16_32_8_1_s							dataflow	0	
isConverged_double_16_16_32_8_1_s		2079	6.929E3		2079		no	0	
writeCenters_double_16_16_32_8_1_s		18795	6.264E4		18795		no	0	
VITIS_LOOP_760_1						300	no		
kMeansTrainIter_double_16_16_32_8_1_s							dataflow	0	
axiVarColToStreams_32_512_64_1		200081	6.670E5		200078		dataflow	0	
convertPort_64_8_16_1_s							dataflow	0	
kMeansPredict_double_16_16_32_8_1_s		1600034	5.333E6		1600034		no	0	
kMeansPredictImp_double_16_16_32_8_1_s		1600033	5.333E6		1600029		dataflow	0	
repeateData_64_16_16_8_1_s		320009	1.067E6		320009		no	0	
repeateData_64_16_16_8_1_Pipeline_V		320004	1.067E6		320004		no	0	
VITIS_LOOP_251_1		320002	1.067E6	4	1	320000	yes		
closestCenter_double_16_16_32_8_1_s		1600028	5.333E6		1600029		dataflow	0	
computingDistance_double_16_16_32_		1600028	5.333E6		1600028		no	0	
computingDistance_double_16_16_3	II Violation	1600024	5.333E6		1600024		no	0	
DISTANCE		1600022	5.333E6	28	5	320000	yes		
minDist_double_16_16_8_s		40014	1.330E5		40014		no	0	
minDist_double_16_16_8_Pipeline_V	Timing Viola	40011	1.330E5		40011		no	0	
VITIS_LOOP_415_2		40009	1.330E5	12	2	20000	yes		
updateC0_double_16_16_32_8_1_s		809154	2.697E6		809154		no	0	
updateC0_double_16_16_32_8_1_Pipeline_V		18	59.994		18		no	0	
VITIS_LOOP_392_1_VITIS_LOOP_393_2		16	53.328	2	1	16	yes		
updateC0_double_16_16_32_8_1_Pipeline_S	II Violation	800011	2.666E6		800011		no	0	
SUM_LOOP		800009	2.666E6	15	5	160000	yes		
AVEG_LOOP		9120	3.040E4	1140		8	no		
AVEG_K		1138	3.793E3	569		2	no		
AVEG_BATCH2		560	1.866E3	35		16	no		

Timeline trace:



The figures below show that the data transferred from DDR to KmeansKernel and back to DDR, and the whole process is done by host program using openCL API.



VI. Reference

https://xilinx.github.io/Vitis_Libraries/data_analytics/2020.1/index.html

https://github.com/Xilinx/Vitis_Libraries/tree/master/data_analytics

Github: Luyee24/HLS_Lab_C (github.com)