Optimization of a Machine Learning Algorithm on the Heterogeneous system using OpenCL

Min Gyung Song, and Dongweon Yoon

***Abstract*—** Today, there is no one who disagrees on how important data is in every industry especially in enterprise market. More recently, the key point that decides the survival of a business is the management of their big data, which is defined by the 3V's: Volume, Velocity, and Variety [1]. While the rate of data generation increases exponentially, processing that data with the limited resources can be a burden to the both business managers and IT managers. Therefore many researchers have already studied new systems which can serve as an alternative resource to calculate and process data.

Parallel hardware, such as a general-purpose GPU (GPGPU), is one of the most well-known alternative. With them, it is possible to process various applications, including data-intensive applications, quickly [2]. OpenCL, in collaborated with several GPU vendors and software organizations, has been launched by the Khronos group as the first open standard platform for the programming of both the GPUs and CPUs [3]. It makes the binary codes execute on various heterogeneous processing units such as CPUs, GPUs and FPGAs simultaneously. It also supports small clients like mobile GPUs for the mobile world.

This paper proposes the method to optimize a machine learning algorithm with the heterogeneous platform which uses both the CPUs and GPUs using OpenCL. Through the experiment, we show that our method can reduce the execution time of the k-means nearest clustering algorithm, which is one of the most common algorithms in the machine learning industry, up to 40%. The more data we use in our system, the faster our results are when compared to the experiment in the multi-core system.

***Keywords*—**Machine Learning, k-means algorithm, OpenCL, Heterogeneous computing

I. Introduction

PROCESSING applications and its data is becoming more important in modern society. There are several ways to process applications and data by using different computing systems. For example, multi-core, many-core, or heterogeneous-core can be used. Above all, a heterogeneous system not only include CPUs but also GPUs or DSPs simultaneously. In this paper, we deal with a heterogeneous computing system which typically include both multi-core CPUs and multi-core GPUs.

OpenCL is the first open standard parallel programming framework for this heterogeneous computing system. It is designed to process large data sets rapidly by allowing a programmer to control the specific hardware resources through OpenCL language. Therefore**,** it can especially be useful in processing large amounts of data very quickly.

We use OpenCL as a method to control our heterogeneous platform including both the CPUs and the GPUs. We also use the k-means nearest clustering algorithm for a machine learning task as a target to optimize in this heterogeneous architecture. The k-means nearest clustering algorithm needs the simple iterative computation logics, therefore it is useful to apply to any machine learning workload. Each iteration group has the input vectors of their nearest center and the mean value of each group, which is then used as the center for the next iteration [4].

The k-means nearest clustering algorithm is a classical clustering algorithm that divides a collection of vectors into k groups. Each vector belongs to the cluster of the nearest centroid. For the machine learning, it can be used when the people don't know the target value or the label as a result of algorithmic processing. Therefore it can be a type of 'unsupervised algorithm' [5].

In this paper, we implemented the k-means nearest clustering algorithm that is optimized to increase performance using OpenCL. Using the experiment's results, we can optimize the algorithm to our heterogeneous system to obtain the best performance. In the section 2, we introduce the concepts of the k-means nearest clustering algorithm briefly. In the section 3 we describe the details of our implementation of the algorithm using OpenCL and also describe the method how we optimized the OpenCL code with this algorithm compared with C code. In the section 4, we present the experimental results to validate our optimized method. At the end of this paper, we give the conclusion in the section 5.

Mingyung Song is a master's student with the Department of Electronics and Computer engineering in the Hanyang University and also with Enterprise and Partner Group in Microsoft Korea, Seoul, Republic of Korea, (e-mail : smkyoyo@hanyang.ac.kr).

Dongweon Yoon, corresponding author, is a professor with the Department of Electronics and Computer engineering in the Hanyang University, Seoul, Republic of Korea. (e-mail : dwyoon@hanyang.ac.kr).

II. K-means Clustering Algorithm

A. Concept of the algorithm

The k-means nearest clustering algorithm is one of the clustering algorithm as shown in the figure 1. In this figure, it is represented that the key concept of the k-means nearest clustering algorithm. Using this algorithm, users can make the k clusters, which bind the most relational points.

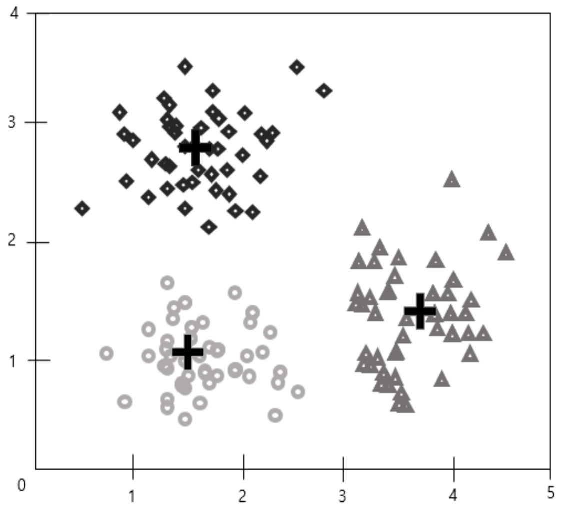


Fig. 1 Result of working with K-means clustering algorithm

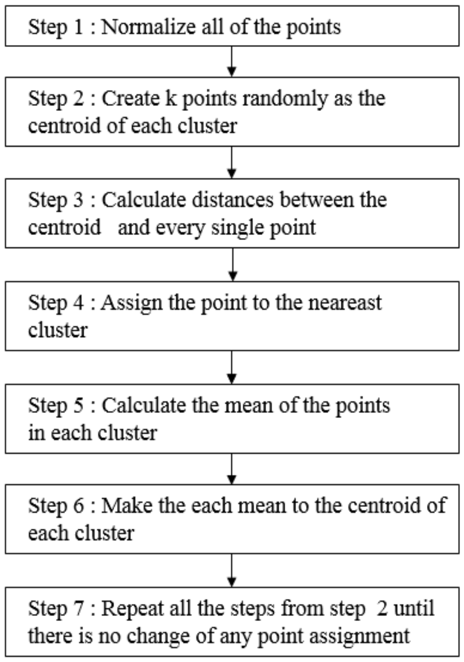
The position of all points in this figure divided by the 3 clusters. The points which have the same shape indicate that they are in the same cluster. The cross shapes indicate the centroid or the mean value of each cluster. It is the basic concept of the k-means nearest clustering algorithm.

B. Processing k-means algorithm

In section B, we present how the k-means nearest clustering algorithm processes its data roughly with 7 steps. In the figure 2, the 7 steps are described.

The first step of the k-means nearest clustering algorithm is normalizing all the points. The experiment in this paper only uses 2 dimensional values. If someone wants to use more dimensions they could do with the same way. This means that according to the range of input data in each dimension, the influence of each dimension can be absolutely different. In order to miniscule this effect fairly, normalization should be conducted to make each dimension has the same influence to the results.

In the step 2, the algorithm randomly creates k-points initially, which are used as the initial values of each cluster's centroid. In the step 3, we calculate the distances between the centroid set from the step 2 and every single point. The step 4 assigns all the points to the nearest cluster from each point. To update clusters more accurately, the algorithm calculates the mean value among the points in each cluster in the step 5.



Step 1 : Normalize all of the points
Step 2 : Create k points randomly as the centroid of each cluster
Step 3 : Calculate distances between the centroid and every single point
Step 4 : Assign the point to the neareast cluster
Step 5 : Calculate the mean of the points in each cluster
Step 6 : Make the each mean to the centroid of each cluster
Step 7 : Repeat all the steps from step 2 until there is no change of any point assignment

Fig. 2 The process of K-means nearest clustering algorithm

After calculating the mean value of each cluster, the values calculated in the step 5 are used as the new centroids of the clusters. In the last place, all the steps starting from the step 2 are repeated until there is no change of assignment.

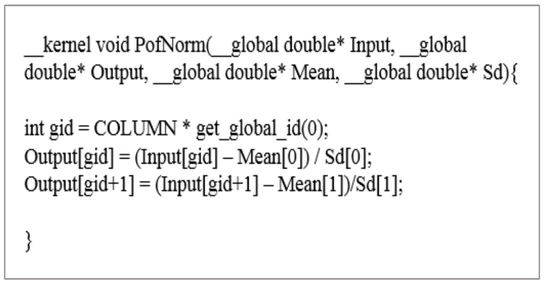
III. OpenCL Implementation

```
for(j=0; j<COLUMN; j++){
                double columnSum = 0.0;
                for(i=0; i<ROW; ++i){
                                colsum += data[i][j];
                }
                double mean = columnSum/ROW;
                double sum = 0.0;
                for(i=0; i<ROW; i++){
                                sum += (data[i][j]-mean) * (data[i][j]-
mean);
                }
                double sd = sum/ROW;
                for(i=0; i<ROW; i++){
                                data[i][j] = (data[i][j] - mean) /sd;
                }
}
```

Fig. 3 A code fragment of the C program

The code in the figure 3 shows a small part of a C code implementation. It is the code fragment associated with the step

1: Normalize all of the points. All tasks in this C code is implemented using the iteration jobs. It is converted into the OpenCL kernel code using the data parallelism method as shown in the figure 4.

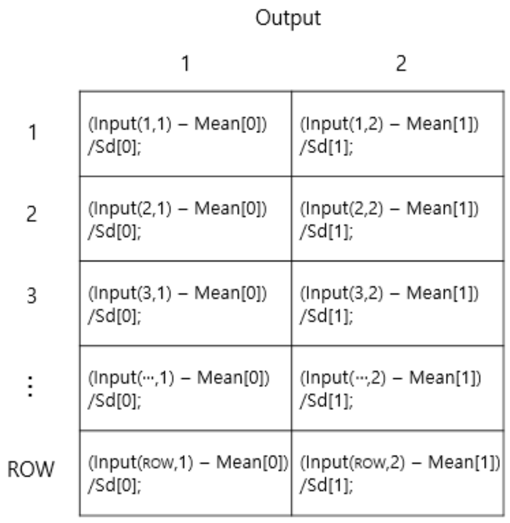


```
__kernel void PofNorm(__global double* Input, __global
double* Output, __global double* Mean, __global double* Sd){

int gid = COLUMN * get_global_id(0);
Output[gid] = (Input[gid] – Mean[0]) / Sd[0];
Output[gid+1] = (Input[gid+1] – Mean[1])/Sd[1];

}
```

Fig. 4 A code fragment of the OpenCL kernel program

The implementation using OpenCL can be divided in two parts. The first part is the OpenCL C code for the CPU cores as the host device. The OpenCL C has some extensions and restrictions in add to fundamentally standard C (C99) [6]. The other part is the kernel code for the GPU cores playing a role as the compute devices.

The code fragment in the figure 4 shows the definition of kernel function 'PofNorm'. It is the part of the kernel code which is applied to GPU cores. This code calculates the normalized values from the raw data only using 3 lines in parallel without any iteration code as shown in the figure 3.



Output

	1	2
1	(Input(1,1) – Mean[0]) /Sd[0];	(Input(1,2) – Mean[1]) /Sd[1];
2	(Input(2,1) – Mean[0]) /Sd[0];	(Input(2,2) – Mean[1]) /Sd[1];
3	(Input(3,1) – Mean[0]) /Sd[0];	(Input(3,2) – Mean[1]) /Sd[1];
⋮	(Input(⋯,1) – Mean[0]) /Sd[0];	(Input(⋯,2) – Mean[1]) /Sd[1];
ROW	(Input(ROW,1) – Mean[0]) /Sd[0];	(Input(ROW,2) – Mean[1]) /Sd[1];

Fig. 5 The result of visualizing PofNorm kernel code

The value of 'gid', which means the global work item size, is set to the number of rows, as same with the number of all the data points in our data set. The reason we only used the data parallelism model using function 'clEnqueueNDRangeKernel' in the host code is because it does the same jobs to every single input points. Therefore, it is inefficient to separate the same task into more than two parts.

The result of visualizing the figure 4 can be presented to the figure 5. It visualizes how the kernel code processes the data to get the normalized values simultaneously.

IV. Experimental Results

The platform used for the performance evaluation in this experiment consisted of an Intel i7- 4790 (3.6GHz) Quad-core processor as the host device, an AMD Radeon R9 290 Hawaii GPU as the compute device, and 16GB of DRAM. In this compute device, the maximum local work item size is restricted to 256 work items per dimension. The physical number of compute units that the operating system can recognize on the compute device is 40.

The GPU has 2560 streaming processors physically. The execution time of the implementation with the OpenCL was compared with that of one implemented in C. The results are shown in the table 1 and the figure 6.

TABLE I
EXECUTION TIME OF THE K-MEANS CLUSTERING ALGORITHM OF C AND OPENCL IMPLEMENTATION

Number of Points	100	1,000	10,000	100,000	1,000,000	2,000,000
C [μs]	295	1,751	11,299	41,666	315,128	629,534
OpenCL[μs]	51,353	98,146	71,637	94,385	166,529	253,858

In this experiments, presented in the table 1, we set the global work item size to the row number in the input data set. The local work item size is set to 256 which is the maximum value of the local work item size in a dimension. The table 1 shows the execution time of the implementation using C code compared to the implementation using OpenCL.

It shows that using C code took less time up to the point where the input data size was 100,000 points. However, when more than 1,000,000 points were used, the execution time for the C implementation exceeded the time that OpenCL took to process the input data. Same result is described as a graph in the figure 6.

In this graph, we can see that the execution time of the OpenCL implementation increase slowly according to increases in the data size. However the execution time of the C implementation increases very rapidly. As a result, the rate of increase of the execution time for the OpenCL implementation is remarkably lower than the rate of increase of the processing time for the C implementation.

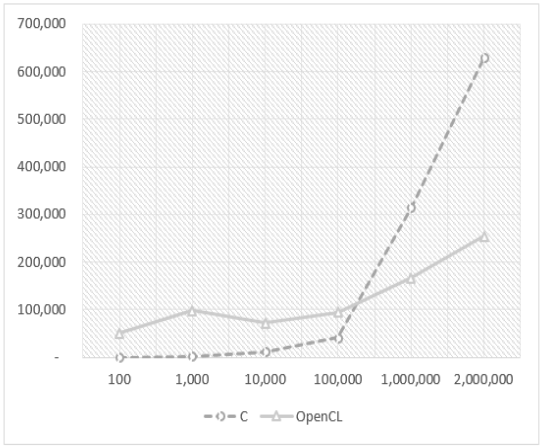


Fig. 6. Execution time of the k-means clustering algorithm of C and OpenCL implementation

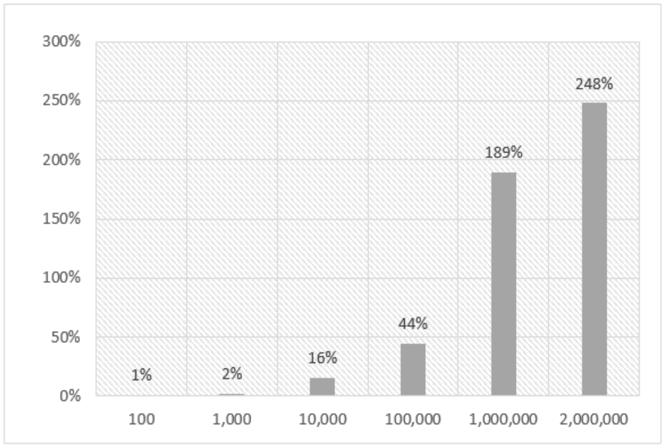


Fig. 7 The rate of execution time C[μs]/OpenCL[μs]

Using the same code, we conducted another experiment using different numbers of compute units, which are presented in the table 2.

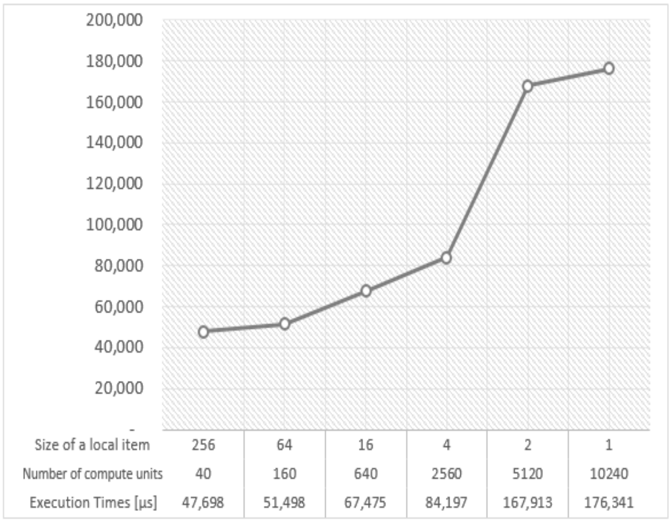
TABLE II
EXECUTION TIME OF THE K-MEANS NEAREST CLUSTERING ALGORITHM WITH RESPECT TO THE NUMBER OF COMPUTE UNITS WITH VARIOUS SIZE

Number of Points	10240	10240	10240	10240	10240	10240	20480
Number of Compute units	40	160	640	2560	5120	10240	80
Size of Local item	256	64	16	4	2	1	256
Times[μs]	47,698	51,498	67,475	84,197	167,913	176,341	195,878

To guarantee the accuracy of the result using the same kernel code, the global work item size should be kept as the row number of the data at all times. As a result, we could only control the number of the compute units and the size of local item at a time.

As shown in the table 2, even though the same input data and algorithm was used, the execution time of each experiment changed according to the size of the local item. It was directly correlated to the number of the compute units because the number of compute units was multiplied by the size of local item in each experiment equaled the number of the points in the data set.

In the table 2, we found that making the number of compute units used in an experiment closer to the physical number of the compute units, in this case 40, made better performance compared to other configurations. The figure 8 shows that the number of the compute units can be a key fact that can decide the performance of the implementation of the algorithm using OpenCL.



Size of a local item	256	64	16	4	2	1
Number of compute units	40	160	640	2560	5120	10240
Execution Times [μs]	47,698	51,498	67,475	84,197	167,913	176,341

Fig. 8 Simulation result of the k-means nearest clustering algorithm with respect to the number of compute units with various size

V. CONCLUSIONS

In this paper, we proposed a method to optimize a machine learning algorithm for a heterogeneous platform using the OpenCL framework. The OpenCL is the first open standard language and framework for general purpose parallel programming for heterogeneous systems.

The machine learning is another big talking point today since its influence in the every industry is growing by the vitalization of big data. In the machine learning, k-means nearest clustering algorithm is one of the representative classifications algorithm, which is very useful for certain types of data in unsupervised environment.

We showed that the performance of the experiment implemented using OpenCL was better than the result of the C code when the size of the input data was sufficiently large in this two dimensional input data set. We confirmed this result in the experiment when the number of the points in the input data set was larger than 1,000,000 points, at which point the ratio of execution times C[μs]/OpenCL[μs] was 189%.

ACKNOWLEDGMENT

This work was partially supported by the MSIP, Korea, under the CITRC support program (NIPA -2014 -H0401 -14 -1001) supervised by the NIPA, and the NRF grant funded by the Korea government (MSIP) (NRF-2014R1A2A1A11052701).

REFERENCES

[1] Laney, Douglas. "3D Data Management: Controlling Data Volume, Velocity and Variety", in *Gartner*, Retrieved 6 Febrary 2001.

[2] Nalia Farooqui, Christopher J. Rossbach, Yuan Yu, "Dynamic Instrumentation and Optimization for GPU Applications", sfma14.cs.washington.edu

[3] Khronos group, https://www.khronos.org/opencl/.

[4] Mihai Budiu, Dennis Fetterly, Michael Isard, Frank McSherry, and Yuan Yu, "Large-Scale Machine Learning Using DryadLINQ", SCALING UP MACHINE LEARNING parallel and distributed approaches, 2012, pp52,

[5] Peter Harrington, *Machine Learning in Action*, *Jpub*, 2013, pp 259—282.

[6] Fixsatrs, The OpenCL Programming book, http://www.fixstars.com /en/opencl/book/OpenCLProgramming Book/opencl-c/