# Highly Parameterized K-means Clustering on FPGAs: Comparative Results with GPPs and GPUs

Hanaa M. Hussain, Khaled Benkrid, Ahmet T. Erdogan
School of Engineering
Edinburgh UniversityEdinburgh
Scotland, U.K.
{h.hussain, k.benkrid, Ahmet.Erdogan}@ed.ac.uk

Huseyin Seker
Bio-Health Informatics Research Group
De Montfort University
England, U.K.
hseker@dmu.ac.uk

***Abstract*—K-means clustering has been widely used in processing large datasets in many fields of studies. Advancement in many data collection techniques has been generating enormous amount of data, leaving scientists with the challenging task of processing them. Using General Purpose Processors or GPPs to process large datasets may take a long time, therefore many acceleration methods have been proposed in the literature to speed-up the processing of such large datasets. In this work, we propose a parameterized Field Programmable Gate Array (FPGA) implementation of the K-means algorithm and compare it with previous FPGA implementation as well as recent implementations on Graphics Processing Units (GPUs) and with GPPs. The proposed FPGA implementation has shown higher performance in terms of speed-up over previous FPGA GPU and GPP implementations, and is more energy efficient.**

***Keywords- FPGA; GPP; GPU; K-means; Microarray;***

## I. INTRODUCTION

Current technologies in many fields of studies have been utilizing advanced data collection techniques which output enormous amount of data. Such data may not be useful in their collected form unless they are computationally processed to extract meaningful results. Current computational power of General Purpose Processors or GPPs has not been able to keep up with the pace at which data are growing [1]. Therefore, researchers have been searching for methods to accelerate data analysis to overcome the limitation of GPPs, one of which is the use of hardware in the form of Field Programmable Gate Arrays (FPGAs).

K-means clustering is one of the widely used data mining techniques to analyze large datasets and extract useful information from them. Previously, we have implemented the K-means clustering on FPGA to target Microarray gene expression profiles, and reported encouraging speed-ups over GPPs [2]. However, the implementation was limited to single dimension and eight clusters only. In this work, we present the implementation of a highly parameterized design which overcomes the previous limitations. Although the proposed design is meant to target Microarray gene expression profiles, it can be adopted for use in other applications such as image segmentation. In this work we compare our parameterized implementation with other FPGA implementation of the K-means algorithms. Furthermore, we will compare the performance of our design with one that has been implemented on Graphics Processing Units (GPUs), a technology that has been gathering a lot of interest in the computing community because of its high performance and relatively low cost.

## II. K-MEANS CLUSTERING

K-means clustering is one of the unsupervised data mining techniques used in processing large datasets by grouping objects into smaller partitions called clusters, where objects in one cluster are believed to share some degree of similarity. Clustering methods helps scientists in many fields of studies in extracting relevant information from large datasets. To arrange the data into partitions, at first one needs to determine the number of clusters beforehand, and initialize centers for each cluster from the dataset. There are several ways for doing this initialization, one way is by randomly assigning all points in the overall dataset to one of these clusters, then calculate the mean of each cluster and use the results as the new centers. Another way is to randomly select cluster centers from the whole dataset. The distance between each point in the datasets and every cluster center is then calculated using a distance metric (e.g. Euclidean, Manhattan). Then, for every data point, the minimum distance to all clusters centers is determined and the point gets assigned to the closest cluster. This step is called cluster assignment, and is repeated until all of the data points have been assigned to one of the clusters. Finally, the mean of each cluster is calculated based on the accumulated values of points in each cluster and the number of points in that cluster. Those means become the new cluster centers, and the process iterates for a fixed number of times, or until points in each cluster stop moving across to different clusters.

The Euclidean metric given in (1) is widely used with K-means clustering and one that results in better solutions [4].

$$D(P,C) = \sqrt{\sum_{i}^{n} (P - C)^2} \quad (1)$$

where P is the data point, C is the cluster center, and n is the number of features. On the other hand, Euclidean distance consumes a lot of logic resources when implemented in hardware due to the multiplication operation used for obtaining the square operation. Therefore, previous groups working on hardware implementation of K-means clustering

978-0-7695-4551-6/11 $26.00 © 2011 IEEE
DOI 10.1109/ReConFig.2011.49

for image segmentation have used the Manhattan distance shown in (2) as an alternative to the Euclidean distance.

$$D(P,C) = \sum_{i}^{n} |P - C| \quad (2)$$

where P is again the data point, C is the cluster center, and n is the number of features. Their results showed that it performed twice as fast as that obtained by Euclidean distance [3]-[9].

Distance computation is the most computationally demanding part, and where most of the K-means processing time occurs. Therefore, accelerating K-means algorithm can be achieved by mainly accelerating the distance computation part, which is achieved using hardware.

## III. REATED WORK ON K-MEANS CLUSTERING ACCELERATION

K-means has already been implemented in hardware by several groups; most were to target applications in hypespectral imaging, or image segmentation. The following review will cover some of the work done on K-means clustering acceleration using FPGA and GPUs.

### *A. K-means clustering acceleration on FPGAs*

In 2000, Lavenier implemented systolic array architecture for K-means clustering [6]. He moved the distance calculation part to FPGA while keeping the rest of the K-means tasks on GPP. The distance computation involved streaming the input through an array of Manhattan distance calculation units of numbers equal to the number of clusters, and obtaining the cluster index at the end of the array. The disadvantage of this approach was the communication overhead between the host and the FPGA. Lavenier tested his design on several processing boards, and one of the relevant speed-ups obtained compared to GPP was 15x [6]-[7].

Between 2000-2003, Leeser *et al.* reported several works related to K-means implementation on FPGA, based on a software/hardware co-design approach [3]-[5]. Their design was partitioned between FPGA hardware and a host microprocessor, where distance calculation and data accumulation were done in hardware in purely fixed-point while new means were calculated in the host to avoid consuming large hardware resources. They achieved a speed-up of 50x over pure GPP implementation. Their design benefited from two things: the first was using Manhattan distance metric instead of the commonly used Euclidean metric to reduce the amount of hardware resources needed, and the second was truncating the bitwidth of the input data without sacrificing accuracy [8].

In 2003, Bhaskaran [9] implemented a parameterized design of the K-means algorithm on FPGA where all the K-means tasks were done in hardware, except the initialization of cluster centers which was done on a host. This design implemented the division operation within FPGA hardware to obtain the new means, using dividers from Xilinx Core Generator. However, this design was tested only on three clusters and achieved a speed-up of 500x over Matlab implementation including I/O overhead [9].

### *B. K-means clustering acceleration on GPUs*

Several implementations of K-means clustering using Graphics Processing Units (GPUs) have been reported in the literature. In 2008, Fairvar implemented K-means on GPU and achieved speed-up of 13.57x (the GPU implementation took 0.724s compared to 9.830s on GPP) when clustering 1 million points into 4000 clusters using NIVIDA GeForce 8600 GT and a 2 GHz GPP host [10]. Another group [11] presented good results when implementing K-means using two types of GPUs. The speed-ups achieved when clustering 200K to 1M on the NVIDIA's GeForce 5900 was between 4x to 12x more than a Pentium 4, 1.5 GHz CPU, and up to 30x when using NVIDIA's GeForce8500 and Pentium 4, 3GHz CPU. They also found that GPU performance was less affected by the size of the dataset as compared to GPPs. Another result reported by this group was related to the effect of the number of clusters on speed-up, where achieved speed-ups were between 10x and 20x for clusters less than 20 using the NVIDIA's GeForce 5900 GPU, and more than 50x when there were more than 20 clusters. For 32 clusters, they reported a speed up of 130x on the NVIDIA's GeForce 8500 GPU.

In 2010, Karch reported a GPU implementation of K-means clustering for accelerating color image segmentation in RGB space [12]. We compared the results published in this work with our FPGA implementation and shall present this comparison in the result section. Choudhary et al reported another GPU implementation using NVIDIA's GeForce 8800GT, which achieved speed-ups between 9x and 40x for datasets ranging from 10,000 to one million points when clustered into 20 clusters [13]. The group also found speed-ups to increase as datasets grow in depth.

## IV. FPGA HARDWARE DESIGN

In this work, a highly parameterized hardware design of the K-means algorithm is presented, which aims to carry all K-means tasks in hardware. The architecture of the whole design consists of a number of blocks which execute kernels within the K-means algorithm. The design generates the required hardware resources and logics based on the parameters entered by the user at compile time. These parameters are the wordlength of the input, number of clusters, number of data points (depth), and dimensions of the input data. The design was captured in Verilog HDL language. Fig. 1 summarizes the main blocks of the design, and in the following sections we shall explain each block in turn. The design performs all the K-means steps within the FPGA, including the division operation and avoids directing any task to an off-chip resource although this capability can be exploited when needed. In the following sections an overview about each block will be presented along with the timing of each block.

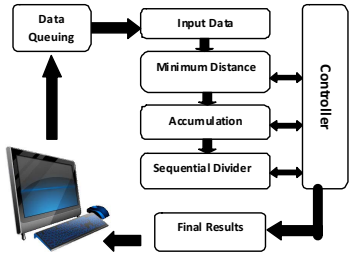


Figure 1. The main blocks of the K-means hardware design

### A. Distance Kernel

This block receives streaming input data stored in on-chip Block RAMs or from an off-chip memory, along with the initialized or updated clusters centers, and computes the distances between each data point and all cluster centers simultaneously. The hardware resources inferred by the synthesis tool to generate multiple distance processors (DPs) is based on the number of clusters chosen, and the detailed logic of every DP is based on the dimensions of the dataset. Each DP is responsible for the computation of the distance between all the input dimensions and one of the cluster centers. Thus one DP is required for each cluster. The DPs work simultaneously such that the distances between every point to all clusters are computed in one clock cycle, hence fully exploiting the parallelism associated with the distance calculation kernel. We have used the Manhattan distance to simplify the computation and save logic resources although other distances could be plugged in instead. The datapath of this block depends on the number of dimensions in the dataset such that the number of stages in the datapath is given by (3):

$$\text{Ceil}\,[\log_2(\text{number of dimensions})] \quad (3)$$

The computation time for a single pass through the whole dataset is a function of the number of data points in the set, which is equal to the depth of the input memory, as shown in (4):

$$\text{Distance Computation Time} = 2\ \text{CLK (latency)} + \text{Memory depth} \quad (4)$$

This kernel has a throughput of 1 data point per clock cycle, and latency of two clock cycles.

### B. Minimum Distance Finder kernel

This block has the role of comparing the outputs of the previous block and determines the minimum distance and the index of the cluster closest to the data point. It consists of a comparator tree, which has stage levels based on the number of clusters, with the number of stages given by (5):

$$\text{Ceil}\,[\log_2(\text{number of clusters})] \quad (5)$$

The number of clusters alone affects the hardware resources inferred by the verilog code, and the datapath of this block is the same as the number of stages needed to complete the comparison. The combined execution time of the above two blocks consisting of the distance computation and minimum distance finder can be summarized in (6):

$$2\ \text{CLK (latency)} + \text{ceil}[\log_2(\text{dimensions})] + \text{ceil}[\log_2(\text{no. of clusters})] \quad (6)$$

### C. Accumulation kernel

This block is responsible for accumulating the data points in the accumulator corresponding to a specific cluster index, and incrementing the corresponding counter, keeping track of the number of points in each cluster along with the values of these points. The block receives the data point under processing along with the index of the cluster having the minimum distance, which was obtained from the previous block, and performs the accumulation and counting accordingly. The number of accumulators inferred by the HDL code is equal to the number of clusters multiplied by the data dimensions:

$$\text{Accumulator numbers} = \text{number of clusters} \times \text{data dimensions} \quad (7)$$

As for the inferred number of counters, it is equal to the number of clusters only, so that each cluster has a counter associated with it. We use fixed point arithmetic with extra care taken in choosing the appropriate wordlength (WL) for the distance, accumulator, and counter results to minimize hardware resources and avoid data overflow. The latter is achieved through error and range analysis of data in all blocks. Based on the input WL selected initially to represent the data, and the number of points in the dataset to be processed, the accumulator and counter WLs were calculated as shown in (8) and (9) respectively:

$$\text{Accumulator WL} = \log_2[(\text{the maximum range that can be represented by the input WL}) \times \text{depth of data}] \quad (8)$$

$$\text{Counter size} = \log_2[\text{depth of the data set}] \quad (9)$$

In the case of Microarray datasets for instance, data usually do not exceed 24,000 points for Human. Hence, 15 bits are found to be sufficient for each counter and 32 bits for each accumulator given the input point is represented by 13 bits. In general, a program was written to automate the range analysis to make the process of sizing easy for any dataset.

### D. Sequentioal Divider Kernel

The Divider kernel is responsible for receiving results from the accumulation kernel and calculating the new cluster centers. The divider itself was generated using Xilinx Core Generator tool, which was found to be faster due to high pipelining when compared to other dividers. In the case of Microarray data, for instance, the generated divider uses 32 bits for the dividend, and 15 bits for the divisor. These values were chosen based on the results of the accumulator/counter blocks as the role of the divider is to divide each accumulator result over the corresponding

counter result to obtain the new cluster centers. Once signaled to start, this block starts scheduling the data received to be serviced by the divider core serially. The number of divisions that needs to be performed is the same as the number of clusters specified in the design parameters. After all the divisions are completed, results are packed to the output port of the divider block. The number of clock cycles taken by the divider to complete its work is a function of the divider latency and the number of clusters as well as the number of dimensions as shown in (10):

$$\text{Divider time} = (\text{core latency}) + (\text{clusters number} \times \text{data dimensions}) \quad (10)$$

In the case of Microarray data, the divider has a latency of 84 clock cycles based on the sizes of the dividend and divisor; however the remaining part will be subject to change as per the initial parameters entered by the user.

The choice of using the pipelined divider from the core generator as opposed to a serial divider is that we have compared the performance of both in terms of area and timing. The serial divider was found to process one bit of the information at a time, thus for a 32 bit dividend and 8 clusters the number of clock cycles that will be needed is 256 clock cycles as compared to 92 for the pipelined divider. This timing difference amplifies as we use more dimensions, for example using 10 dimensions and 8 clusters causes the serial divider to take 2560 clock cycles while the pipelined divider takes only 164 clock cycles. On the other hand, the number of slices consumed by the serial divider is a lot less than the pipelined, where the latter consumes 1389 slices compared to 91 slices for the serial divider, when using Virtex 4 FPGA. Since the aim of the project is to accelerate the K-means algorithm, the pipelined divider was favored.

## V. IMPLEMENTATION RESULTS

Each block in the design was implemented and tested on the Xilinx ML403 platform board, which has a XC4VFX12 FPGA chip. The board runs at a clock frequency of 100 MHz. The design was captured in Verilog, simulated, synthesized, placed and routed using Xilinx ISE 12.2 tool. Finally, a bitstream file was generated and downloaded to the board for testing. In the following sections, performance results of our design, as well as comparison with GPP, FPGA, and GPU implementations are presented.

### *A. Comparison with GPP*

Implementation results when clustering a dataset of 2905 points and one dimension, with input wordlength of 13 bits to 8 clusters, showed that the complete design consumed 2985 slices (740 CLBs), and achieved a maximum clock frequency of 142.8 MHz. When comparing the runtime of this hardware implementation with a software version running on an Intel Core 2 Duo 3.0 GHz CPU with 3 GB RAM, running Windows XP Professional operating system, the following speed-up results were obtained as shown in Table I below.

TABLE I. PERFORMANCE RESULTS

| Software (ms) | Hardware (ms) | Speed- up |
|---|---|---|
| 6.1 | 0.723 | 8.4x |

### *B. Comparison with another FPGA design*

In general, it is difficult to compare like for like FPGA implementations because of the use of different FPGA families and chips, as well as different design parameters. Nonetheless, we have attempted a comparison here with the closest FPGA implementation to ours, namely the one reported in [3]. Here, we compare our parameterized core design excluding the divider to make it compatible with the FPGA implementation reported in [3] which performed the division operation on a host. Both implementations were based on data size of 1024 × 1024 with 10 dimensions, 12 bits data, and 8 clusters. In both cases, data were stored off chip and streamed on to the FPGA. Comparative results are shown in Table II below. Obviously, our implementation is faster because it is based on a more recent FPGA technology, but it is also more compact using normalized slice/LUT count. More importantly, it is more flexible as it has a higher degree of parameterization compared to the implementation reported in [3].

TABLE II. IMPLEMENTATION RESULTS

| Compare | Xilinx XCV1000 [3] | Xilinx XC4VFX12 |
|---|---|---|
| **Slices** | 8884 (out of 12288) | 5107 (out of 5549) |
| **LUTs** | 17768 | 10216 |
| **Max Clock Frequency** | 63.07 MHz | 100 MHz |
| **Single loop processing time** | 0.17 s | ~ 0.07 s |

### *C. Comparison with GPUs*

When comparing the performance of our FPGA K-means clustering implementation to a recent GPU implementation presented in [12] for an image processing application, we found that the FPGA solution outperforms the GPU in terms of speed as shown in Table III. The results shown are based on two different datasets, one is 0.4 Mega Pixel (MPx) in size and the other is 6.0 MPx. Both datasets were processed for 16, 32, and 64 clusters, and both were for a single dimension. Both GPP and GPU results were based on an Intel Core 2 Duo 2.2 GHz, with 4GB memory and Nvidia GeForces 9600M GT graphics card, running Microsoft Windows 7 Professional 64-bit. On the other hand, the targeted FPGA device was Xilinx XC4VSX35, the design used 13 bits to represent the dataset, and could run at a maximum clock frequency of 141 MHz. The Virtex device we used in this comparison is not a high end FPGAs, the latter can achieve higher speeds but we tried to limit the choice to a reasonable size that can accommodate the design and be reasonable for comparing with the above GPU. Both of the images were too large to be stored within the FPGA, therefore, off chip memory was needed to store data which

TABLE III. EXECUTION RESULT OF K-MEANS IN GPP, FPGA, GPU, FOR SINGLE DIMENSION DATA

| Clusters | GPP Avg time per iteration (sec) [12] | GPP Avg time for complete execution (sec)[12] | GPU Avg time per iteration (sec) [12] | GPU Avg time for complete execution (sec)[12] | FPGA per iteration (sec) | FPGA Complete execution (sec) |
|---|---|---|---|---|---|---|
| **0.4 MPx** | | | | | | |
| 16 | 0.269 | 4.314 | 0.021 | 0.443 | 0.0028 | 0.0392 |
| 32 | 0.516 | 7.637 | 0.020 | 0.421 | 0.0028 | 0.042 |
| 64 | 1.004 | 12.78 | 0.023 | 0.508 | 0.0028 | 0.0454 |
| **6 MPx** | | | | | | |
| 16 | 4.279 | 67.07 | 0.256 | 5.176 | 0.0425 | 0.723 |
| 32 | 8.144 | 110.7 | 0.247 | 4.439 | 0.0425 | 0.638 |
| 64 | 15.86 | 208.2 | 0.270 | 5.220 | 0.0425 | 0.723 |

were streamed on to the FPGA pixel by pixel, and one data point was read every clock cycle. The processing times reported in [12] do not include the initialization of clusters centers and the input/output stage, and similarly with the FPGA times reported in Table III.

The speed-up results of our FPGA implementation over the GPP and GPU results reported in [12] are shown in Fig. 2 below. From the FPGA/GPP curve, it is clear that there is a linear relationship between the number of clusters and the achieved acceleration. Similar observation was found when comparing GPU with GPP. Both observations confirm that FPGA and GPU outperform GPP as the number of clusters is increased. On the other hand, the FPGA/GPU curve shows that FPGA outperforms GPU in terms of execution time; this is due to higher exploitation of parallelism in FPGA. However, the FPGA/GPU acceleration is not greatly affected by the number of clusters (up to 64 clusters in our experiments) as found with GPP. As for the device utilization, the XC4VSX35 FPGA used in this comparison has 15360 slices, which was enough to implement the logic required to accommodate the number of clusters shown in Table III. With the 16 clusters, the implementation occupied 5177 slice (33%), and with 32 and 64 clusters, the implementation occupied 8055 slices (52%) and 13859 (98%) respectively.

In addition, the effect of data dimensionality on performance of both FPGA and GPU implementations was investigated in this work. In [14], authors reported results of clustering Microarray Yeast expression profiles, where they have achieved a speed-up of 7x to 8x over GPP for 4 and 9 dimensions respectively. The size of the dataset was 65,500, and the original dimensions were 79. We compared our FPGA implementation with the GPU in [14] for the same size and dimensions of the data used in [14] and found that FPGA achieves speed-up between 2x to 7x over GPU. Table IV details these results based on XC4VSX35 FPGA, and Nvidia 8600 GT GPU. The drop in performance in GPU as the dimensions increased could be due to the way the implementation utilizes resources within the GPU when computing specific kernels. Furthermore, we include in the table expected speed-up results of a multi-core FPGA implementation of the K-means which we have proposed previously in [2]. The multi-core approach can clearly outperform the GPU for problems requiring small or reasonable number of clusters and dimensions that can be mapped easily on to commercially available FPGA devices.

Moreover, the power and energy consumption of the three K-means implementations were compared and reported in Table V. Both GPP and FPGA power figures were actually measured, while the GPU power figure was obtained from the Nvidia GeForce 9600 GT datasheet, reflecting the power rating of the device [15]. The results in Table V are based on using 13 bits to represent the 0.4 MPx image case shown in Table III, and targeting the Xilinx Virtex XC4VFX12 FPGA available on the ML 403 board we used with the image being stored in off-chip memory.

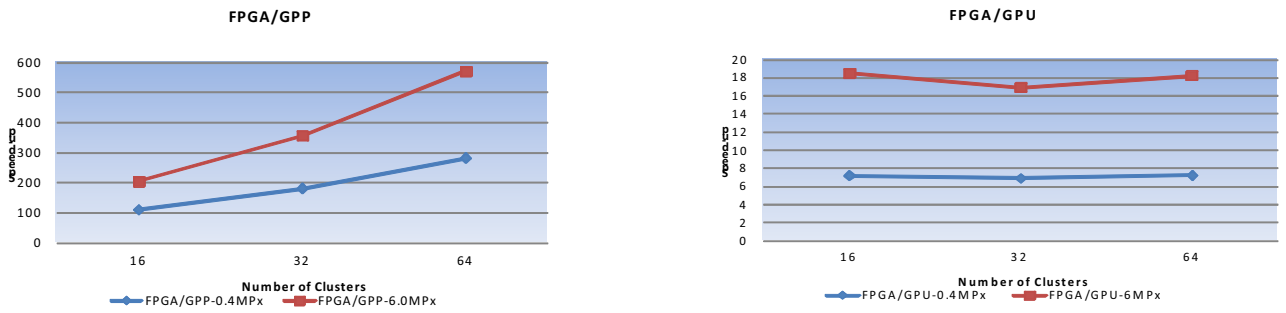


Figure 2. Speed-up results of the FPGA implementation of K-means over both GPP and GPU implementation

TABLE IV. EXECUTION RESULTS OF K-MEANS IN GPP, FPGA, GPU FOR MULTIDEIMENSIONAL DATA

| Clusters<br>DIMENSIONS=4 | GPP time per iteration (sec) [14] | GPU time per iteration (sec) [14] | GPU/GPP Speed-up | FPGA time per iteration (sec) | FPGA/GPP Speed-up | FPGA/GPU Speed-up | 5 cores FPGA/GPU |
|---|---|---|---|---|---|---|---|
| 3 | 0.0495 | 0.00623 | 8x | 0.0019 | 26x | 3.2x | 16x |
| 4 | 0.0652 | 0.00902 | 7.2x | 0.0042 | 15.5x | 2x | 10x |
| **DIMENSIONS=9** | | | | | | | |
| 3 | 0.1031 | 0.0125 | 8.2x | 0.0019 | 54x | 6.7x | 33.5x |
| 4 | 0.1333 | 0.01589 | 8.4x | 0.0042 | 31.7x | 3.8x | 19x |

TABLE V. COMPRISON OF POWER AND ENERGY OF DIFFERENT K-MEANS IMPLEMENTATIONS

| Platform | Power (Watt) | Execution time for the 0.4MPx image, with 16 clusters (sec) | Energy (Joule) |
|---|---|---|---|
| **GPP** | 120 | 4.314 | 517 |
| **GPU** | 59 | 0.443 | 26 |
| **FPGA** | 15 | 0.056 | 0.84 |

From Table V, we can see that FPGA is ~8x more power efficient than GPP, and ~4x more power efficient than GPU. In addition, the FPGA implementation is 615x more energy efficient than the GPP, and ~31x more energy efficient than the GPU. This implementation utilized 4909 slices (89%) of the targeted device.

## VI. COCLUSION AND FUTURE WORK

The design and implementation of a highly parameterized FPGA core for K-means clustering was presented in this paper. This outperformed equivalent GPP and GPU implementations in terms of speed (two orders and one order of magnitude respectively). In addition, the FPGA implementation was more energy efficient than both GPP (615x) and GPU (31x). This makes the FPGA implementation highly desirable although FPGAs still suffer from a relatively higher cost of purchase and development compared to GPPs and GPUs. The lack of standard API tools and hardware boards is a big contributor in this.

Future work will harness dynamic partial reconfiguration to exploit FPGA resources more efficiently. We also plan to implement other data mining techniques on FPGAs including kNN and SVM to specifically target Microarray data.

## ACKNOWLEDGMENT

We would like to thank the “Public Authority of Applied Education and Training” in Kuwait for sponsoring this study and to also thank “Kuwait Foundation for the Advancement in Sciences” for its contribution to this study.

## REFERENCES

[1] P. Kumar, B. Ozisikyilmaz, W. Liao, G. Memik, and A. Choudhary, “High Performance Data Mining Using R on Heterogeneous Platforms,” *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE Int.Symp.on*, 2011, pp.1720-1729.

[2] H. Hussain, K. Benkrid, H. Seker, A. Erdogan, “FPGA Implementation of K-means Algorithm for Bioinformatics Application: An Accelerated Approach to Clustering Microarray Data,” *in Proc. 2011 Adaptive Hardware and Systems (AHS), NASA/ESA Conf.*, 2011, pp.248-255.

[3] M. Leeser, P. Belanovic, M. Estlick, M. Gokhale, J.J. Szymanski, and J. Theiler, "Applying reconfigurable hardware to the analysis of multispectral and hyperspectral imagery*," in Proc. SPIE*, 2002, pp.4480.

[4] M. Estlick, M. Leeser, J. Theiler, and J.J. Szymanski, "Algorithmic Transformations in the Implementation of K- means Clustering on Reconfigurable Hardware," *in Int. Symp. on Field Programmable Gate Arrays*, 2001, pp. 103-110.

[5] M.D. Estlick, "An FPGA Implementation of the K-means Algorithm for Image Processing", M.S. thesis, Dept. Elect. Eng., Northeastern Univ., Boston, MA., 2002.

[6] D. Lavenier, "FPGA implementation of the K-means clustering algorithm for hyperspectral images," Los Alamos National Laboratory, LAUR # 00-3079, pp. 1-18, 2000.

[7] M. Gokhale, J. Frigo, K. McCabe, J. Theiler, C. Wolinski, and D. Lavenier, "Experience with a hybrid processor: K-means clustering," J. Supercomputing, vol. 26, pp. 131-148, 2003.

[8] J. Theiler, M.E. Leeser, M. Estlick, and J.J. Szymanski, "Design Issues for Hardware Implementation of an Algorithm for Segmenting Hyperspectral Imagery,” *in Proc. Imaging Spectrometry VI*, vol. 4132, 2000, pp. 99-106.

[9] V. Bhaskaran, "Parametrized Implementation of K-means Clustering on Reconfigurable Systems," M.S. thesis, Dept. Elect. Eng., Univ. of Tennessee, Knoxville, TN, 2003.

[10] R. Farivar, D. Rebolledo, E. Chan, and R. H. Cambell, “A parallel implementation of k-means clustering on GPUs,” in *Proc. 2008 Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 08)*, Las Vegas, NV,2008, pp.340-345.

[11] S. A. Shalom, M. Dash, and M. Tue, “Efficient kmeans clustering using accelerated graphics processors,” *in Proc. 10th Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK '08)*, 2008, pp. 166–175.

[12] G. Karch, "GPU based acceleration of selected clustering techniques," M.S. thesis, Dept. Elect. and Comp. Science, Silesian University of Technology in Gliwice, Silesia, Poland, 2010.

[13] A. Choudhary, D. Honbo, P. Kumar, B. Ozisikyilmaz, S. Misra and G. Memik, “Accelerating Data Mining Workloads: current approaches and future challenges in system architecture design,” *Wiley Interdisciplinary Reviews: WIREs Data Mining and Knowledge Discovery*, vol.1, pp.41-54, January/February 2011.

[14] S. A. Shalom, M. Dash, and M. Tue, “GPU-based fast k-means clustering of gene expression profiles,” Presented at the 12th Annu. Int. Conf. on Research in Computational Molecular Biology (RECOMB), Singaphore, 2008.

[15] Nvidia GEForce 9600 GT datasheet at: http://www.nvidia.com/object/product_geforce_9600gt_us.html [online]