A Fast and Scalable FPGA-Based Parallel Processing Architecture for K-Means Clustering for Big Data Analysis

Ramprasad Raghavan and Darshika G. Perera
Department of Electrical and Computer Engineering, University of Colorado at Colorado Springs, Colorado, USA
email: darshika.perera@uccs.edu

Abstract—**The exponential growth of complex, heterogeneous, dynamic, and unbounded data, generated by a variety of fields including health, genomics, physics, climatology, and social networks pose significant challenges in data processing and desired speed-performance. Existing processor-based software-only algorithms are incapable of analyzing and processing this enormous amount of data, efficiently and effectively. Consequently, some kind of hardware support is desirable to overcome the challenges in analyzing big data. Big data analytics involves many important data mining tasks including clustering, which categorizes the data into meaningful groups based on the similarity or dissimilarity among objects. In this research work, we introduce an efficient FPGA-based parallel processing architecture for K-means Clustering, one of the most popular clustering algorithms. Experiments are performed on a benchmark dataset to evaluate the feasibility and efficiency of our hardware design. Our hardware architecture is generic, parameterized, and scalable to support larger and varying datasets as well as a varying number of clusters. Our hardware configuration with 32 processing elements (PEs) achieved 368 times speedup compared to its software counterpart.**

Keywords-Big data analysis; parallel processing architecture; FPGAs; K-Means clustering; embedded hardware; hardware algorithms; data mining

I. INTRODUCTION

Since the late-2000s, data acquisition techniques and data storage media have evolved rapidly. This has resulted in an exponential growth of complex, heterogeneous, dynamic, and unbounded data being generated by a variety of fields including health, genomics, physics, climatology, and social networks. For instance, in genomics, the amount of sequence data generated doubled every seven months within the last decade, now producing several peta-bytes of data every year [20]. Also, the volume of data currently produced by NASA Earth science mission is about 12 peta-bytes, and is expected to grow by an order of magnitude within the next five years [4]. Analyzing and processing such an enormous amount of data pose serious challenges to the data mining community.

Big data analytics often involves many important data mining tasks such as [15]: classification, clustering, regression, and association rule mining. From these, we are focusing on the most widely used clustering and classification. Most of today's data mining tasks, including clustering and classification, for big data analysis are becoming more complex (compute/data intensive), requiring more processing power than ever before. Also, in many cases, the data needs to be processed in real-time to yield the actual benefit. These constraints have a significant impact on the speed-performance of the data mining applications.

Existing algorithms for big data analytics are typically processor-based (software-only) designs. These processor-based algorithms are incapable of analyzing and processing enormous amounts of data, efficiently and effectively. A survey done in [19] demonstrated that processor-based computing platforms, including multi-processor, multi-core, GPGPU (General Purpose Graphics Processing Unit) are simply not sufficient to handle this enormous amount of data. Consequently, new design techniques, architectures, and computing platforms are needed to overcome the challenges in analyzing big data.

In order to satisfy the constraints and requirements associated with big data analytics, it is imperative to provide some kind of hardware support. In this research work, we investigate special-purpose hardware for big data analysis. Special-purpose or customized hardware is optimized for a specific application and avoids the high execution overhead of fetch/decode/execute instructions as in processor-based software-only designs [3]. As a result, customized hardware provides higher speed-performance, lower power consumption [6],[9] and area-efficiency compared to the equivalent software running on a general-purpose processor.

Our main objective is to provide efficient hardware architectures for big data analysis to satisfy the associated constraints and requirements. In this work, we focus on hardware support for clustering techniques in data mining, specifically K-Means Clustering, one of the most popular clustering algorithms.

We make the following contributions in this paper.

- We introduce a novel and efficient embedded architecture for K-Means Clustering for data mining. Our proposed architecture is generic, parameterized and scalable. Our design can process varying data sizes (i.e., any number of vectors (D) and any number of attributes (N)) and varying number of clusters (K). Our design can be configured to have varying number of parallel PEs (P) to further enhance the speed-performance.
- We introduce an efficient "Data Engine" to pre-fetch the essential data (for processing) from the off-chip external memory to the on-chip memory, thus significantly reducing the memory access latency.

978-1-5386-0700-8/17/$31.00 ©2017 IEEE

- We use a register-based interface and industry standard AXI-bus, which would enable seamless integration of our design to other computing platforms and systems.
- We design embedded software for K-Means Clustering to evaluate our embedded hardware design.
- We implement different hardware configurations with varying number of parallel PEs. We perform experiments on these configurations with varying data sizes and with varying number of clusters. We analyze the timing, speed-performance, and resource utilization for each configuration.

II. Existing Research Work on Hardware Support for K-Means Clustering

We investigated existing research work on hardware support for K-Means Clustering algorithm.

Hardware-software co-design solutions were proposed in [7],[13],[14], specifically for spectral and hyper-spectral image analysis applications. In this case, the host-processor off-loads only the compute-intensive Distance Measure to a dedicated hardware co-processor on a different platform, thus incurring high communication overhead between the two processors.

In [17], a hardware design was proposed for a modified version [11] of the K-Means Clustering. This design can only process small datasets, and can only function efficiently when the clusters are distributed far apart.

Parameterized hardware designs for K-Means Clustering were proposed in [1],[10] using Manhattan Distance. The hardware designs claimed to achieve higher speedup compared to their software counter-part. However, the proposed hardware designs and the software-only designs were executed on different platforms and on different processors. For [1], the comparison was with MatLab simulation. In addition, it is designed to process only a fixed number of vectors and a fixed number of clusters [10], although the design is claimed to be parameterized.

In [12], a parallel hardware design was proposed, where Manhattan Distance was executed in parallel. The design achieved a speedup of 3, compared to the existing FPGA-based hardware designs for the K-Means Clustering. However, the proposed design was implemented on an advance Virtex-6 FPGA, whereas designs used for comparison purposes were implemented on older Xilinx FPGAs. Since the CMOS process technology of an FPGA has a significant impact on the frequency and the occupied area of the design, these comparisons are not necessarily fair. Furthermore, system-level designs were not proposed, which is essential when processing a large volume of data that typically exists in many data mining techniques.

The above investigations revealed several issues in the existing hardware designs for the K-Means Clustering: Manhattan Distance is used instead of Euclidian Distance, which lowers the accuracy of the Distance Measure [5]; integer division was used, which also affects the accuracy of both the Distance and Centroid Measure computations; proposed hardware designs and their software counter-parts were executed on different platforms and different processors, thus the performance comparisons are not necessarily fair; most proposed designs, except [1], lack the industry standard protocols for communicating between the host-processor and the FPGA; most of these designs were implemented for specific applications, accordingly the datasets and the parameters are fixed, thus can not be modified for other applications with different parameters.

In this research work, we propose a novel and efficient hardware architecture for the K-Means Clustering algorithm, while addressing the aforementioned issues in the existing designs.

III. Design Approach and Development Platform

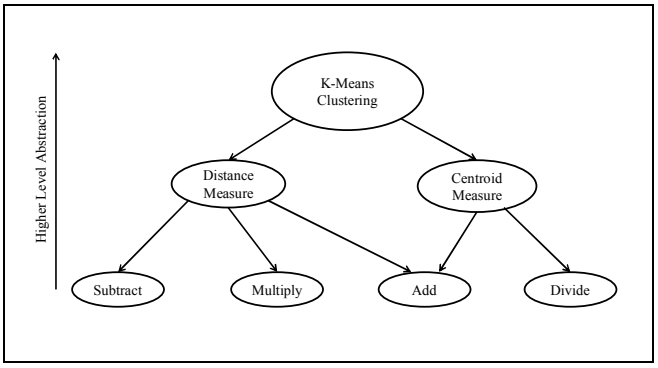

Figure 1. Hierarchical Platform-Based Design Approach.

In our designs, both software and hardware versions of various operations are implemented using a hierarchical platform-based design approach to facilitate component reuse. As depicted in Fig. 1, our design consists of different levels of abstractions, where higher-level functions utilize lower-level sub-functions and operators.

The hardware modules for the adder and the subtractor are designed using Verilog, while the multiplier [24] and the divider [23] are selected from the Xilinx IP core library. The fundamental hardware modules are designed using integer operators, except the division, which uses a fixed-point divider. The fundamental software modules are designed in similar fashion. The MicroBlaze is configured to use a floating-point unit to perform the division operation.

All our hardware and software experiments are carried out on the Xilinx ML605 development platform. This platform utilizes a Xilinx Virtex-6 XC6VLX240T-1FFG1156 FPGA, which consists of MicroBlaze soft processors, 37680 slices, 2MB of BRAMs (Block Random Access Memory), and 512MB DDR3-SDRAM (Double-Data-Rate Synchronous Random Access Memory) external memory (to hold large volume of data). The ML605 has several external non-volatile memories including: 128MB of Platform Flash XL, 32MB BPI Linear Flash, and 2GB Compact Flash, to hold the configuration bitstreams.

Our customized hardware modules are designed in mixed VHDL and Verilog. They are executed on the Virtex-6 FPGA (running at 100MHz) for correctness and performance verification. Xilinx ISE 14.7 and XPS 14.7 are used for the hardware designs. ModelSim SE and Xilinx ChipscopePro 14.7 are used to verify the results and functionalities of the

designs. Our software modules are written in C and executed on the MicroBlaze soft processor (running at 100MHz) on the same FPGA (for fair comparison purposes) with level II optimization. Xilinx XPS 14.7 and SDK 14.7 are used to verify the software modules.

A user-designed hardware counter is used to measure the execution time in clock cycles for both the hardware and software designs. The performance gain or speedup resulting from the use of hardware over software is computed using the following equation:

$$Speedup = \frac{BaselineExecutionTime(Software)}{ImprovedExecutionTime(Hardware)} \quad (1)$$

During the initial design and development phase, we used a small synthetic data set (consisting of 10 of 2-attribute vectors) to perform our experiments. K is considered as 4, thus having 4 clusters. The results are manually evaluated and compared with the experimental results for both the hardware and software designs to verify the correctness.

For the remaining experiments, we decide to use a real benchmark dataset, particularly designed for clustering. After investigating several database archives, we decide on a benchmark dataset for "Wholesale Customer Data" [2] from the UCI Machine Learning Repository. This dataset has 440 records (or vectors) of integer data, each having 8 attributes. The dataset represents the clients of a wholesale distributor. It provides the annual customers' spending on six different product categories from two different channels.

A. System-Level Architecture

Fig. 2 illustrates the system-level interfacing for our hardware and software designs for the K-Means Clustering. Since the 2MB on-chip memory in Virtex-6 FPGA is not sufficient to store a large volume of real data found in many data mining applications, we integrated a 512MB DDR3-SDRAM [25] into the system. DDR3-SDRAM and the DDR3-SDRAM memory controller run at 200MHz, while the rest of the system is running at 100 MHz. As shown in the Fig. 2, the AXI (Advanced Extensible Interface) acts as the glue logic for the whole system.

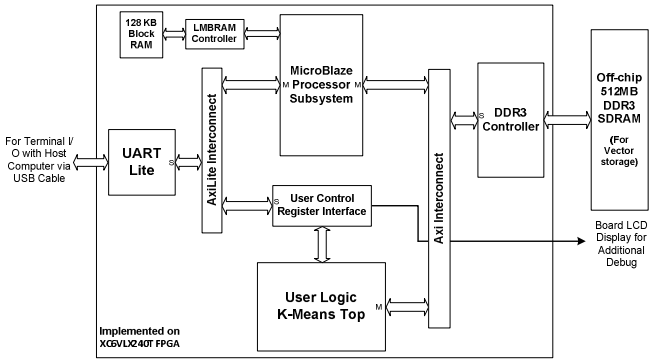

Figure 2. System-Level Architecture Block Diagram.

We partitioned our hardware architecture (i.e., user-designed module in Fig. 2) into two separate modules: K-Means Top and K-Means Register Interface. The K-Means Register Interface module provides necessary information to the K-Means Top module, including the specific addresses to access the DDR3-SDRAM, and the required amount of data to be transferred.

In order to communicate with the MicroBlaze and the DDR3-SDRAM, the user-designed hardware module is connected to the AXI4 bus [21] through the AXI Intellectual Property Interface (IPIF) module, using a set of ports called Intellectual Property Interconnect (IPIC). With this AXI system-level interface, the user-designed hardware can receive a signal from the MicroBlaze and start processing, read/write data/results from/to the SDRAM, and send a signal to the MicroBlaze when the execution is completed.

The MicroBlaze processor [25] also communicates with other peripherals via AXI bus. The processor is configured to have 128KB for the Instruction and Data Caches, which is the minimum on-chip memory, required to processes the software designs.

From our previous work on hardware support for data mining applications [16], it was observed that a significant amount of time was spent on accessing DDR3-SDRAM external memory, which is a major performance bottleneck. For the current system-level design, we incorporated several techniques, including AXI burst and pre-fetching techniques, to reduce the memory access latency. In this case, the user-designed hardware is enhanced with burst reads/writes from/to SDRAM through the IPIF module. In addition, the data for specific computation is pre-fetched by the Data Engine to an internal pre-fetch buffer (Fig. 4 in Section IV). The pre-fetching technique is discussed in Section IV.

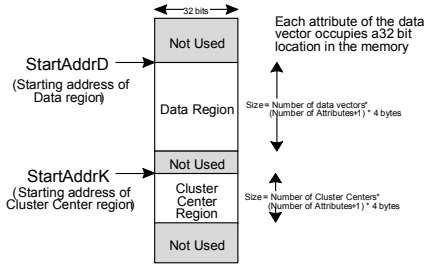

Figure 3. DDR3-SDRAM Memory Organization.

For our designs, as illustrated in Fig. 3, we partitioned the DDR3-SDRAM memory into two regions: Data Vector and Cluster Center. The MicroBlaze processor maps these regions to non-overlapping memory locations. The processor initializes the Cluster Center region by randomly selecting a required number of vectors from the Data Vector region. The final Cluster Centers are updated in the Cluster Center region after the completion of the K-Means Clustering algorithm.

IV. EMBEDDED HARDWARE ARCHITECTURE FOR K-MEANS CLUSTERING

In this section, we introduce our embedded hardware architecture for the K-Means Clustering algorithm. The hardware architecture for the K-Means Top module, as shown in Fig. 4, consists of several sub-modules: Data

Engine, Data Pre-fetch Memory, Cluster Memory, Distance Measure computation, and Centroid Measure computation.

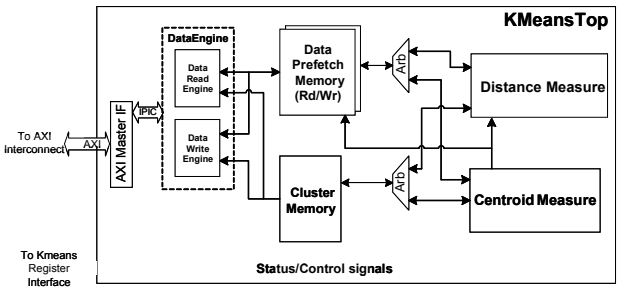

Figure 4. K-Means Top Module.

The Data Engine is responsible for all the data transfers between the user-designed hardware and the DDR3-SDRAM. The data fetched from the SDRAM is aligned with the internal data structure and buffered to the internal pre-fetch memory. Instead of single reads/writes, AXI burst is used for data transfer to reduce the memory access latency between the user-designed hardware and the SDRAM.

The Data Pre-fetch Memory and Cluster Memory are designed using the BRAMs [22] from the Xilinx IP core library. The Data Pre-fetch Memory has one buffer to read data and another buffer to write data.

Firstly, the initial Cluster Centers are fetched from the Cluster Center region of the DDR3-SDRAM, and stored in the internal Cluster Memory. Secondly, the K-Means Clustering is performed. The K-Means Clustering algorithm consists of 2 major steps. The first step is to compute the Distance Measure, and the second step is to compute the Centroid Measure. This is an iterative process, where these two steps are repeated. At the end of each iteration, the interim Cluster Centers are re-evaluated and stored in the internal Cluster Memory. This iterative process continues until a specified or a maximum number of iterations is reached or until the Clusters Centers converge to local minima. Next, the final Cluster Centers are written to the Cluster Center region of the DDR3-SDRAM from the internal Cluster Memory.

After the execution of the K-Means algorithm, each vector in the data set belongs to a specific Cluster Center. The number of vectors per cluster (defined by a Cluster Center) is also specified in the Cluster Center region of the DDR3-SDRAM. This clustering information can be used for subsequent data mining analysis such as Data Abstraction.

A. Distance Measure Hardware Design

For our hardware design, we selected the Euclidian Distance over other Distance Measure computations, due to its accuracy [5]. The origianl equation for the Euclidian Distance is as follows:

$$d(p,q) = \sqrt{\sum_{k=1}^{n} |p_k - q_k|^2} \qquad (2)$$

where n is the total number of attributes; p_k and q_k are the k^{th} attribute of the two vectors p and q respectively.

We modified the above equation slightly, and replaced the resource intensive square-root function with the multiplier function, in order to minimize the on-chip hardware resources, while retaining the accuracy of the Euclidian Distance computation.

$$d(p,q)^2 = \sum_{k=1}^{n} |p_k - q_k|^2 \qquad (3)$$

As depicted in Fig. 5, our embedded hardware design for the Distance Measure computation consists of a data path and a control path (CP). The data path of a single processing element (PE) of the Euclidian Distance Measure is designed in a pipelined fashion. As illustrated, the data path consists of a subtractor, a multiplier, an accumulator, and a comparator. The accumulator is designed as a sequence of an adder and an accumulator register with the feedback loop to the adder.

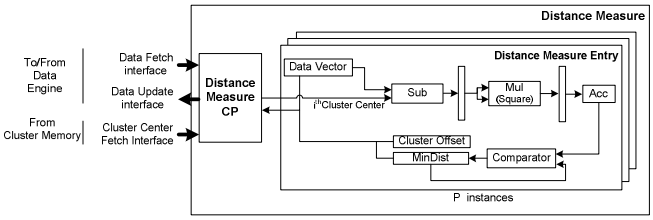

Figure 5. Euclidian Distance Measure Module.

A single PE of the Distance Measure computes the distance between a Cluster Center and a vector, one distance computation at a time. For each vector, the nearest Cluster Center is found by computing the Euclidian Distance between that vector and the existing Cluster Centers. Then the corresponding vector is associated with the nearest Cluster Center by updating the internal Cluster Memory.

In our design, the final Distance Measure, i.e., one distance computation, is the last squared term computed. From equation (3), this is the final accumulation results of the square of the difference in attributes of the vectors. In each iteration, a new final squared distance is compared with the former minimum squared distance. If it is less than the former, the cluster center offset and the new minima are updated for a particular vector. This process continues until all the vectors are processed and assigned to the nearest Cluster Center.

In order to exploit the inherent parallelism in the Distance Measure computation, we employ a parallel processing architecture. We use multiple PEs to perform multiple Euclidian Distance computations in parallel. The number of distance computations processed in parallel depends on the number of Distance Measure PEs (i.e., number of vectors processed in parallel, P) on the chip at a time. Our hardware design is parameterized; by configuring the parameter P, the number of parallel PEs can be varied, during the implementation.

B. Centroid Measure Hardware Design

Our Centroid Measure hardware is designed according to the following equation:

$$C_i = \frac{\sum_{k=1}^{m} d_k}{m} \quad (4)$$

where d is a data vector in the i^{th} cluster and m is total number of vectors in the i^{th} cluster.

As illustrated in Fig. 6, the embedded hardware design for the Centroid Measure computation also consists of a data path and a control path (CP). As shown, the data path of the Centroid Measure consists of an accumulator (Sum), several fixed-point dividers, and several rounding logic. A rounding logic is employed to increase the accuracy of the Mean computation of the Centroid Measure module.

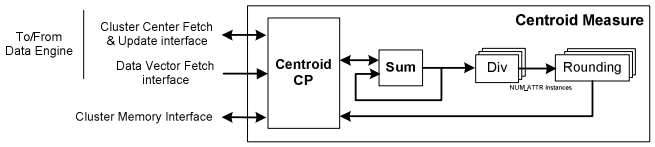

Figure 6. Centroid Measure Module.

Before executing the K-Means Clustering algorithm, the Centroid Measure module fetches the initial Cluster Centers from the Cluster Center region of the DDR3-SDRAM, and stores them in the internal Cluster Memory. During the execution of the algorithm, this module computes the new Cluster Centers, which are considered as the Centroids of the existing clusters. Then the new Cluster Centers are updated to the internal Cluster Memory. After the execution of the algorithm, the Centroid Measure module writes the final Cluster Centers to the Cluster Center region in the DDR3-SDRAM from the internal Cluster Memory.

The Centroid Measure computes new Cluster Centers as the Mean of all the vectors belonging to a particular cluster. In our design, from equation (4), the numerator is computed for an associated attribute of each vector (belonging to the same cluster or Cluster Centers), and only the final accumulation result goes through the divider. We use 8 dividers and 8 rounding logic, one per each attribute. Thus, the Mean computation for all the attributes, in our case 8 attributes, for a particular cluster is processed in parallel.

C. Design Parameters

Our hardware architecture for the K-Means Clustering algorithm is generic and parameterized. It can be generalized to any data mining applications that utilize this algorithm by configuring the parameters listed in Table I. The size of the dataset to be process is configured by the following parameters: the total number of vectors (D), the total number of attributes per vector (N), the number of bits of the attribute (W), and the total number of cluster centers or clusters (K). These variables can be changed externally without changing the underlying hardware architecture. In addition, during the implementation, our hardware architecture can be configured to incorporate varying number of parallel PEs (by configuring the parameter P), in order to process the Distance Measure computation in parallel. As illustrated in Table I, these parameters directly impact the dimensions (width and depth) of the Cluster Memory and the Data Pre-fetch Memory of our design.

TABLE I. DESIGN PARAMETERS

Parameter	Default Value	Description
N	8	Total number of attributes per vector
W	20	Size of each attribute in number of bits
K	64	Total number of clusters for the dataset
D	8192	Total number of vectors in the dataset
P	8	Number of vectors computed in parallel for Distance Measure computation
K_W	Derived	Width of Pre-Fetch memory to hold the entire vector in one line of memory: $(N^* W) + \log_2(D)$
K_MEM_W	Derived	Width of Cluster Memory to hold Mean value of all the vectors within a cluster: $N^*(\log_2(D)+ W) + \log_2(D)$

V. EXPERIMENTAL RESULTS AND ANALYSIS

Experiments are performed on the "Wholesale Customer Data" [2] benchmark dataset to evaluate our embedded hardware architecture for the K-Means Clustering algorithm. The dataset consists of 440 8-attribute vectors; hence the data size is 3520. It should be noted that our design can be used to process any data set regardless of the size of the data set.

For these experiments, the data are pre-fetched from the DDR3-SDRAM to the on-chip BRAM [22], processed, and some of the intermediate results are also stored in the BRAM, and the final results are written back to the SDRAM.

The experiments are performed using various data sizes in order to examine the scalability. The number of attributes and number of bits of the attributes are kept the same and only the data size or the number of vectors is varied. Experiments are also performed with varying number of clusters in order to examine its effect on the performance-gain for a given dataset.

To further examine hardware advantages, a parallel processing architecture is employed. Multiple PEs are used to perform multiple Euclidian Distance computations in parallel. Experiments are performed using four different hardware configurations with varying number of PEs: 1PE, 8PEs, 16PEs, and 32PEs.

A. Execution Times for Embedded Hardware and Software

In order to evaluate the performance of our embedded hardware designs, we design and implement embedded software for the K-Means Clustering algorithm. The software design is executed on the MicroBlaze processor on the same development platform.

The execution times for both the hardware and software designs are obtained using a hardware timer. The execution time is measured in clock cycles, which is a standard unit; hence can be used to estimate the time/speedup of the K-Means Clustering performed on different platforms.

The execution times for four different hardware configurations (Hw) and the software design (Sw) for different data sizes (i.e., varying number of vectors) and for varying number of clusters are presented in Tables II -V. The execution times for 8, 16, 24, and 32 clusters are presented in Tables II, III, IV, and V respectively. The number of

iterations taken for the K-Means Clustering algorithm to converge is presented in column 3.

Since a normal graph would not give us much insight into the execution characteristics, a logarithmic graph is used for the execution times for the K-Means Clustering algorithm for both the hardware and software designs for varying number of clusters.

TABLE II. EXECUTION TIMES FOR FOUR HARDWARE CONFIGURATIONS AND SOFTWARE ON MICROBLAZE FOR 8 CLUSTERS

Data Size	No. of Vectors	No. of Iterations	Execution Time in clk_cycles				
			Sw	Hw (1PE)	Hw (8PE)	Hw (16PE)	Hw (32PE)
352	44	9	2818971	122766	22469	15641	14209
704	88	9	4402706	244274	41047	28956	25047
1056	132	9	5978004	365566	61439	42349	37472
1408	176	14	11735755	757620	124386	85110	77035
1760	220	19	19240848	1284089	211747	143125	129208
2112	264	24	28542706	1945471	316447	216287	196355
2464	308	15	20455800	1418280	231861	157665	142427
2816	352	18	27674706	1945048	315243	213589	194979
3168	396	21	35902646	2552242	415507	279801	256439
3520	440	26	49004681	3510962	567993	386277	352386

TABLE III. EXECUTION TIMES FOR FOUR HARDWARE CONFIGURATIONS AND SOFTWARE ON MICROBLAZE FOR 16 CLUSTERS

Data Size	No. of Vectors	No. of Iterations	Execution Time in clk_cycles				
			Sw	Hw (1PE)	Hw (8PE)	Hw (16PE)	Hw (32PE)
352	44	4	2444617	101949	16709	10456	8740
704	88	6	5708487	303082	45187	28740	20272
1056	132	9	11578280	680025	102230	62597	43686
1408	176	10	16237960	1006633	147401	87237	60678
1760	220	22	43125868	2765596	408583	239538	160199
2112	264	16	36759982	2413223	351006	209224	141089
2464	308	18	47439978	3166746	464218	274632	180442
2816	352	17	50485709	3417651	495818	289587	192196
3168	396	22	72694419	4974745	726383	422578	281730
3520	440	22	80088142	5527472	800692	470977	309363

TABLE IV. EXECUTION TIMES FOR FOUR HARDWARE CONFIGURATIONS AND SOFTWARE ON MICROBLAZE FOR 24 CLUSTERS

Data Size	No. of Vectors	No. of Iterations	Execution Time in clk_cycles				
			Sw	Hw (1PE)	Hw (8PE)	Hw (16PE)	Hw (32PE)
352	44	4	3616875	149021	23303	13883	11070
704	88	8	11253387	590302	83689	51235	33534
1056	132	13	24818972	1435659	206313	121605	80507
1408	176	8	19249929	1177882	164821	93444	61586
1760	220	16	46479140	2941797	415988	233957	146122
2112	264	16	54481907	3529233	490881	281528	178450
2464	308	21	81981553	5402688	758299	431816	264794
2816	352	25	110011991	7349814	1020043	571595	352201
3168	396	20	97948258	6614384	924968	516768	322250
3520	440	15	80938575	5512239	764317	432538	264069

TABLE V. EXECUTION TIMES FOR FOUR HARDWARE CONFIGURATIONS AND SOFTWARE ON MICROBLAZE FOR 32 CLUSTERS

Data Size	No. of Vectors	No. of Iterations	Execution Time in clk_cycles				
			Sw	Hw (1PE)	Hw (8PE)	Hw (16PE)	Hw (32PE)
352	44	4	4795571	196096	29912	17269	13394
704	88	5	9323831	486366	67341	40400	25355
1056	132	15	37890866	2179917	305839	176379	113144
1408	176	12	38286260	2324429	317090	175206	111603
1760	220	12	46173694	2904480	401305	220334	132315
2112	264	17	76723876	4935507	670260	375952	230509
2464	308	14	72414016	4741358	650454	362451	214129
2816	352	14	81625616	5418353	734624	402163	238540
3168	396	18	116780861	7836239	1070727	584575	352475
3520	440	33	235818153	15959500	2160999	1196031	703255

Fig. 7 illustrates the execution times versus the data sizes (number of vectors) for different hardware configurations (with varying number of PEs) and software on MicoBlaze for 32 clusters, whereas the Fig. 8 shows the execution times versus the number of iterations for different hardware configurations (with varying number of PEs) and software on MicoBlaze for 32 clusters. The top line of the graphs indicates the execution times for software on MicroBlaze, whereas the bottom line indicates the execution time for hardware configuration with 32 PEs.

From Fig. 7 and Fig. 8, it can be observed that the execution time for the K-Means Clustering increases, but not linearly, for different hardware configurations (with varying number of PEs) and for software on MicroBlaze. This is expected, since the execution time depends not only on the size of the data but also on the number of iterations taken for the K-Means to converge. The graphs for 8, 16, and 24 clusters show similar behavior.

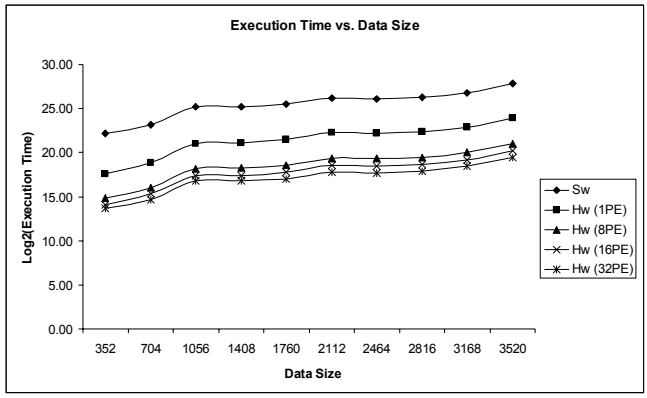

Figure 7. Graph of Execution Time vs. Data Size for 32 Clusters.

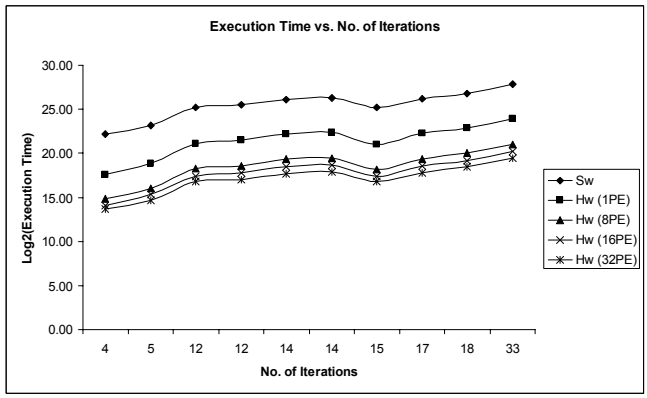

Figure 8. Graph of Execution Time vs. No. of Iterations for 32 Clusters.

B. *Performance Comparison: Embedded Hardware vs. Software on MicroBlaze*

1) For a Constant Number of Clusters: Although the execution times are obtained using varying number of clusters, to illustrate the performance gain of different hardware configurations (with varying number of PEs) for varying data sizes (number of vectors), only the results from the 32 clusters are used. As shown in Fig. 9, the top line of the graph indicates the performance gain for 32PEs, while the bottom line indicates the gain for 1PE.

From Table V and Fig. 9, for a constant number of clusters, the average speedups for 1PE, 8PEs, and 16PEs are 17, 121, and 216, respectively. The average speedup for 32PEs is 343. As expected, the speedup increases with the increasing number of PEs, however, it does not change proportionately with the size of the data. This is because the execution time, hence the speedup, depends not only on the

data size but also on the number of iterations. The graphs for 8, 16, and 24 clusters show similar behavior.

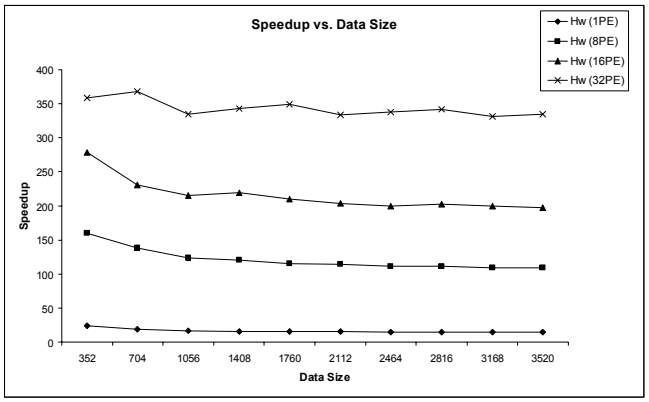

Figure 9. Graph of Speedup vs. Data Size for 32 Cluster.

2) For a Constant Data Size: To illustrate the performance gain of the four embedded hardware configurations (with varying number of PEs) for varying number of clusters, only the results of the largest dataset of size 3520 is used. The execution times and the corresponding speedups for varying number of clusters are presented in Table VI.

TABLE VI. PERFORMANCE COMPARISON: EMBEDDED HARDWARE VS. SOFTWARE ON MICROBLAZE FOR VARYING NUMBER OF VECTORS

No. of Clusters	Execution Time in clk_cycles					Speedup (Sw vs. Hw)			
	Sw	Hw (1PE)	Hw (8PE)	Hw (16PE)	Hw (32PE)	1PE	8PE	16PE	32PE
8	49004681	3510962	567993	386277	49004681	14	86	127	139
16	80088142	5527472	800692	470977	80088142	14	100	170	259
24	80938575	5512239	764317	432538	80938575	15	106	187	307
32	235818153	15959500	2160999	1196031	235818153	15	109	197	335

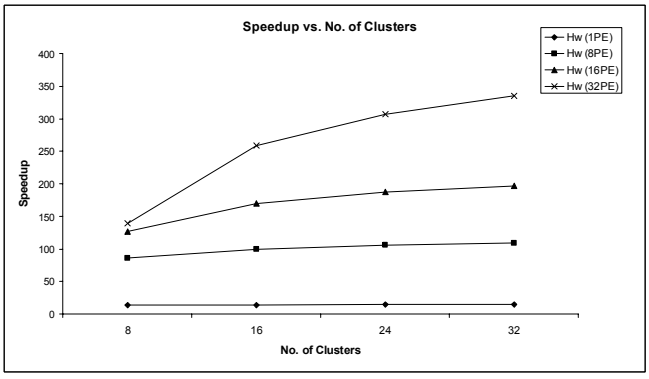

Figure 10. Graph of Speedup vs. Number of Clusters for Largest Data Set.

The Fig. 10 shows the speedup versus the number of clusters for different hardware configurations (with varying number of PEs) for the largest data size of 3520. The top line of the graph indicates the performance gain for 32PEs, while the bottom line indicates the performance gain for 1PE.

For a constant data size and for a constant number of PEs: From Table VI and Fig. 10, for a constant data size, by using the same hardware configuration (with a constant number of PEs) the speedup increases with the increasing number of clusters. For instance, for hardware configurations with 1PE, 8PEs, and 16PEs, the speedups increase by 7%, 27%, and 55% respectively as the number of clusters increases from 8 to 32. For hardware configuration with 32PEs the speedup increases by 141% as the number of clusters increases from 8 to 32.

For a constant data size and for a constant number of clusters: From Table VI and Fig. 10, for a constant data size and for constant number of clusters (in this case 32 clusters), the speedup increases significantly by using a parallel processing architecture. By using 8PEs versus 1PE the speedup increases by 627%; by using 16PEs versus 1PE, the speedup increases by 1213%; by using 32PEs versus 1PE, the speedup increases by 2133%.

As observed from Fig. 10, for 1PE, the rate of increase of speedup is relatively low for increasing number of clusters. The rate of increase of speedup significantly enhances with the increasing number of PEs. The rate of increase of speedup is the highest for 32PEs. Therefore, it can be concluded that for increasing number of clusters and for a constant data size, the performance gain improves, and the incremental rate of performance improvement increases with the increasing number of PEs.

The above experimental results and analysis illustrate that by increasing the number of PEs, to compute the Distance Measure operation in parallel, significantly enhanced the speed-performance of the K-Means Clustering algorithm. This demonstrates that the parallelism as well as the pipeline nature in computations can be exploited to a great extent in hardware.

C. Analysis on Resource Utilization

In order to investigate the feasibility of our embedded hardware designs for different applications on different platforms, cost analysis on space is carried out.

TABLE VII. SPACE STATISTICS FOR VARIOUS HARDWARE CONFIGURATIONS

Configuration	Slice Logic Utilization			
	Number of Occupied Slices	Number of DSP481s	Number of Slice Registers	Number of Slice LUTs
1PE	6916	88	14132	15009
8PE	8140	102	16171	19366
16PE	9820	118	18650	23741
32PE	12377	150	23412	33963

From Table VII, considering the total number of occupied slices (column 2) for each hardware configuration, additional space required to process 8PEs, 16PEs, and 32PEs are 18%, 42%, and 79%, respectively, compared to the space required for 1PE. The increase in occupied area is due to the number of PEs employed to execute the Distance Measure computation in parallel. Therefore, it only affects the hardware footprint of the user-design hardware module, whereas the hardware footprint of the rest of the system remains the same for all four hardware configurations.

Compared to the total number of occupied slices (37680) in this specific FPGA, our hardware configuration with 32PEs only occupies 33% of the chip. This demonstrates that we can utilize the remaining parts of the chip to incorporate more parallel PEs to further enhance the speed-performance.

From Table VII, we could distinguish and predict the resource requirements for hardware configurations with varying number of parallel PEs. Also, from the performance

comparison in Section V.B, we could predict the speedup for different hardware configurations with varying number of parallel PEs. These speed-space tradeoffs can be used to estimate a near optimal configuration for a given system.

Since our hardware architecture is parameterized, the designer has the flexibility to build a faster design with larger footprint, or build a slower design that has a comparatively smaller footprint. The designer should carefully weigh-in the speed-space tradeoffs when determining the most suitable hardware configuration for a specific application as well as for a specific hardware platform.

VI. Conclusion and Future Work

In this paper, we introduced novel and efficient hardware architecture for K-Means Clustering algorithm for big data analysis. We addressed various issues identified in the existing hardware designs for the K-Means Clustering in Section II.A. Our hardware design is generic, parameterized, and scalable. It can be used to process varying data sizes (any number of vectors and any number of attributes) and varying number of clusters for different applications and for different hardware platforms. Our design can also be configured to modify the number of parallel PEs to enhance the speed-performance. Our hardware configuration with 32PEs is executed up to 368 times faster than its software counterpart. We also incorporated several techniques to reduce the memory access latency, which was a major performance bottleneck in our previous work on hardware support for data mining [16].

These experimental results are encouraging and indeed show a great potential in implementing algorithms for big data analysis such as K-Means Clustering using FPGA-based parallel hardware. Currently, we are investigating ways to integrate these novel and efficient embedded architectures to larger systems, including big data centers and genomic sequencing centers, to reduce the computation burden and increase the efficiency of these systems.

References

[1] Bhaskaran, V., "Paramterized Implementation of K-Means Clustering on Reconfigurable Systems," Knoxville, 2003.

[2] Cardoso, M., Wholesale customer Data Set, University of California, Irvine.

[3] Compton, K. and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software", ACM Computing Surveys, vol.34, no.2, pp.171-210, June 2002.

[4] Earley, S. "Really, Really Big Data: NASA at the Forefront of Analytics", IEEE Computing Edge, May 2016.

[5] Estlick, M., M. Leeser, J. Theiler and J. J. Szymanski, "Algorithmic Transformations in the Implementation of K-means Clustering on Reconfigurable Hardware," in ACM/SIGDA 9th Int. Symp. on Field Programmable Gate Arrays, 2001.

[6] Garcia, P., K. Compton, M. Schulte, E. Blem and W. Fu, "An Overview of Reconfigurable Hardware in Embedded Systems", EURASIP Journal on Embedded Systems, pp.1-19, 2006.

[7] Gokhale, M., J. Frigo, K. McCabe, J. Theiler and L. Dominique, "Early Experience with a Hybrid Processor: K-Means Clustering," in 1st Int. Conf. on Engineering of Reconfigurable Systems and Algorithms, 2001.

[8] Hauck, S., and A. Dehlon, Reconfigurable Computing: The Theory and Practice of FPGA-Based Computing, Morgan Kaufmann Publishers, 2008.

[9] Hussain, H.M., K. Benkrid, H. Seker and A. T. Erdogan, "FPGA Implementation of K-means Algorithm for Bioinformatics Application: An Accelerated Approach to Clustering Microarray Data," in NASA/ESA Conf. on Adaptive Hardware and Systems, 2011.

[10] Hussain, H.M., K. Benkrid, H. Seker and A. T. Erdogan, "Highly Parameterized K-means Clustering on FPGAs: Comparative Results with GPPs and GPUs," in Reconfigurable Computing and FPGAs, . 2011.

[11] Kanungo, T., D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman and A. Y. Wu, "An Efficient k-Means Clustering Algorithm:Analysis and Implementation," IEEE Transcactions on Pattern Analysis and Machine Intelligence, vol. 24, pp. 881-892, July 2002.

[12] Kutty, J.S., Farid Boussaid, Abbes Amira, "A High Speed Configurable FPGA architecture for K-mean clustering," IEEE Int. Symp. on Circuits and Systems, pp. 1801-1804, 2013.

[13] Lavenier, D., "FPGA implementation of the k-means clustering algorithm for hyperspectral images," Los Alamos National Laboratory LA-UR #00-3079, 2000.

[14] Leeser, M., P. Belanovic , M. Estlick , M. Gokhale , J. J. Szymanski and J. Theiler, "Applying Reconfigurable Hardware to the Analysis of Multispectral and Hyperspectral Imagery," in SPIE, The Int. Society for Optical Engineering , 2001.

[15] Manning, D.C., Raghvan, P., and Schutze, H., "Introduction to Information Retrieval", Cambridge University Press, 2008.

[16] Perera, D.G., and K.F. Li, "FPGA-Based Reconfigurable Hardware for Compute Intensive Data Mining Applications", In Proc. of 6th IEEE Int. Conf. on P2P, Parallel, Grid, Cloud and Internet Computing, pp.100-108, Oct. 2011.

[17] Saegusa, T., and M. Tsutomu, "An FPGA Implementation of K-Means Clustering for Color Images Based on KD-Tree," in Field Programmable Logic and Applications, 2006.

[18] Salton, G., and M.J. McGill, "Introduction to Modern Information Retrieval", McGraw-Hill, New York, 1983.

[19] Singh, D., and C. Reddy, "A survey on platforms for big data analytics," Journal of Big Data, 2014.

[20] Stephens, Z.D., S.Y. Lee, F. Faghri, R. Campbell, C. Zhai, M. Efron, S. Sinha, R. Iyer and R. Gene, "Big Data: Astronomical or Genomical?," PLoS Biol, 2015.

[21] Xilinx Inc., "LogiCORE IP AXI Interconnect (v1.06.a)," Dec. 2012. http://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v1_06_a/ds768_axi_interconnect.pdf

[22] Xilinx Inc., "LogiCORE IP Block Memory Generator v7.3," 2012. http://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v7_3/pg058-blk-mem-gen.pdf

[23] Xilinx Inc., "LogiCORE IP Divider Generator v4.0," June 2011. http://www.xilinx.com/support/documentation/ip_documentation/div_gen/v4_0/ds819_div_gen.pdf

[24] Xilinx Inc., "LogiCORE IP Multiplier v11.2," Mar. 2011, http://www.xilinx.com/support/documentation/ip_documentation/mult_gen_ds255.pdf

[25] Xilinx Inc., "MicroBlaze Processor Reference Guide," Oct. 2013. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/mb_ref_guide.pdf

[26] Xilinx Inc., "Virtex-6 FPGA Memory Interface Solutions" Mar. 2011, http://www.xilinx.com/support/documentation/white_papers/wp469-microblaze-for-cost-sensitive-apps.pdf.