HIGH SPEED DOCUMENT CLUSTERING IN RECONFIGURABLE HARDWARE

G. Adam Covington, Charles L.G. Comstock, Andrew A. Levine, John W. Lockwood, Young H. Cho

Applied Research Laboratory, Washington University One Brookings Drive, Campus Box 1045
St. Louis, MO 63130-4899 USA
{gac1, aal1, lockwood, young}@arl.wustl.edu, cc1@cse.wustl.edu
`http://www.arl.wustl.edu/projects/fpx/reconfig.htm`

ABSTRACT

High-performance document clustering systems enable similar documents to automatically self-organize into groups. In the past, the large amount of computational time needed to cluster documents prevented practical use of such systems with a large number of documents. A full hardware implementation of K-means clustering has been designed and implemented in reconfigurable hardware that clusters 512k documents rapidly. This implementation, uses a cosine distance metric to cluster document vectors that each have 4000 dimensions. The system was synthesized on a Xilinx XC4VLX200 with a clock frequency of 250 MHz. With this FPGA the hardware accelerated algorithm runs up to 328 times faster than the software version running on an Intel 3.6 GHz Xeon. Experiments were also performed using the Field Programmable Port Extender (FPX) platform. It is shown that a fully pipelined architecture running on a Xilinx XCV2000E-8 FPGA (with a clock frequency of 80 Mhz) can outperform software implementations running on an Intel 3.60 GHz PC by a factor of twenty-six.

1. INTRODUCTION

As the amount of information generated from worldwide sources increases, humans can no longer keep up with the task of reading and categorizing the data. To organize data, a document clustering system can be utilized to automatically group related content together. To cluster high dimensional vectors, microprocessor systems inefficiently perform comparisons sequentially. The algorithms used for clustering have large amounts of computations that can be run in parallel. These parallel algorithms can be implemented in hardware to achieve high performance. Field Programmable Gate Arrays (FPGAs) can be utilized to implement clustering algorithms at high speeds. FPGAs also allow parameters of the clustering algorithms to be modified through reprogramming. Application Specific Integrated Circuits (ASIC) would also allow algorithmic speedup, however they lack the ability to be re-optimized for the parameters of an algorithm. In this paper, we demonstrate a complete system that uses K-means clustering to automatically group related documents together that is implemented in hardware using FPGAs.

1.1. Related Work

Duda and Hart [1] described the original K-means algorithm. Estlick et. al.[2] and Leeser et. al.[3] have used K-means with the Minkowski distance metric L_1 to cluster hyperspectral images using a hybrid software, hardware approach. They compared the use of different Minkowski distance metrics, Manhatten Distance (L_1), Euclidean Distance (L_2) and Max Distance (L_∞), but determined that L_1 was the one that would fit best for their hardware. However, these distance metrics are not well suited for clustering sparse, high dimensional data.

For clustering documents with high dimensionality, the cosine theta distance metric is preferred. The cosine theta distance or spherical distance intuitively provides better distance in high dimensions than euclidian distance.

2. K-MEANS CLUSTERING OF DOCUMENTS

2.1. Concept

Given a set of N documents, D, where each document $\vec{d}$ is described by L dimensions or bins. Each bin counts the frequency of words and similar words map to the same bin. Documents are partitioned into K clusters by minimizing the distance between each document and the cluster centroid. The “centroid” of a cluster, is the mean of all documents in a cluster. This is an approximate representation of all documents in that cluster. Figure 1 shows three documents being mapped into document vectors. These document vectors and their assignments are then used to calculate the cluster centroids.

This research was sponsored by the Air Force Research Laboratory, Air Force Materiel Command, USAF, under Contract Number MDA972-03-9-0001. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFRL or the U.S. Government.

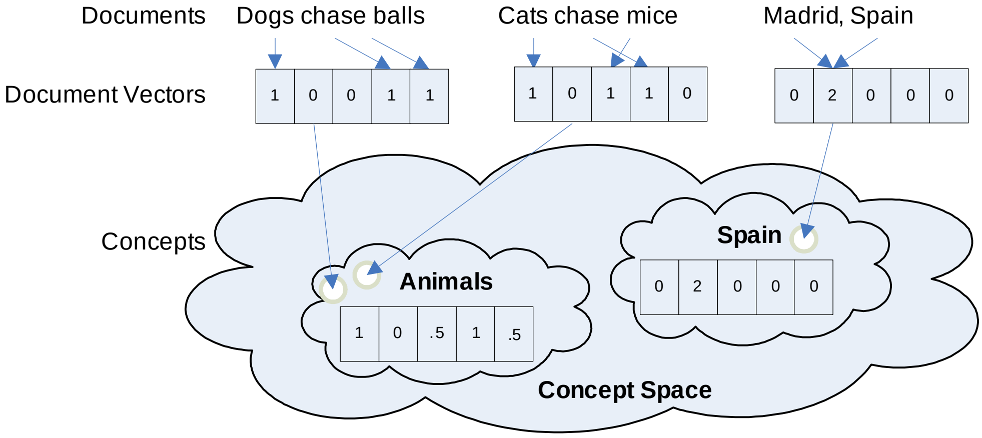

Fig. 1. Documents mapped into L dimensional vectors are clustered into groups of related concepts

2.2. Algorithm

In our case, K-means will minimize the cosine distance between each document $d \in D$ and the cluster centroid C_k for which it is assigned.

$$\sum_{k=1}^{K} \sum_{d \in C_k} \frac{\vec{d} \cdot \vec{C_k}}{|\vec{d}| \cdot |\vec{C_k}|}$$

The K-means algorithm operates as follows:

1. Assign document vectors to a cluster using an initial seed.
2. Initialize cluster centroids from initial document assignments.
3. For each document $d \in D$
   (a) Recalculate distances from document $\vec{d}$ to centroids, and find the closest centroid C_{min}.
   (b) Move document $\vec{d}$ from current cluster C_k into new cluster C_{min} and recalculate the centroid for C_k and C_{min}.
4. Repeat step 3 until either the maximum epoch limit is reached or an epoch passes in which no changes in document assignments are made. An epoch is a complete pass through all documents.

For this implementation of K-means the initial seed clusters were generated by randomly assigning documents to starting clusters. K-means is commonly calculated using double precision arithmetic, however for a hardware implementation, reduced precision integer representations were utilized to increase the speed of the circuit and to fit the design into the available space on an FPGA. Effects of limited precision arithmetic on document classification accuracy are shown in Section 4.3.

2.2.1. Centroid Update

Cluster centroids are initialized by averaging across all of the document vectors in each cluster. For recalculating the centroids after a document update, we add or subtract the document vector from the unscaled centroid dimension $\vec{C}_{unscaled}$ and then average the centroid into the scaled centroid dimension $\vec{C}_{scaled}$ for distance comparison. The average is not necessarily calculated by dividing by the absolute number of documents in the cluster C_{count}, but by a scaled approximation of it.

$$\vec{C}_{scaled} = \frac{\vec{C}_{unscaled}}{C_{count}}$$

However in order to do this calculation in integer arithmetic it is necessary to rescale the centroid and the document count. In our implementation we use the following equation.

$$\vec{C}_{scaled} = \min\left(255, \frac{2 \cdot \vec{C}_{unscaled}}{\min(C_{count}/16, 1)}\right) \quad (1)$$

We use of the minimum function to avoid divide by zero from under scaled document counts, as well as ensuring that the final scaled value fits within the 8 bits allocated per centroid dimension.

2.2.2. Cosine Theta Distance Metric

In order to find the distance between a document and a centroid and thus the closest centroid to a document, the cosine theta distance metric is used. The Cosine Distance metric is defined as follows. Given a document vector $\vec{D}$, and a centroid vector $\vec{C}$, the spherical distance between $\vec{D}$ and $\vec{C}$ is defined as:

$$\cos(\theta) = \frac{\vec{D} \cdot \vec{C}}{|\vec{D}| \cdot |\vec{C}|} \quad (2)$$

Cosine theta distance D is ranged $D \in [0, 1]$. However, in order to represent the cosine distance as integers it was necessary to scale cosine theta into a larger range to approximate fixed point arithmetic.

The integer representation of the equation:

$$\cos(\theta) = \frac{16\left(\frac{16(\vec{D}\cdot\vec{C})}{|\vec{C}|}\right)}{|\vec{D}|} \tag{3}$$

To allow for further speedup by reducing the number of divisions necessary[1]:

$$\cos(\theta) = \frac{16(\vec{D}\cdot\vec{C})}{\min((|\vec{C}|\cdot|\vec{D}|)/16, 1)} \tag{4}$$

The constant multiple of 16 was chosen to rescale the range of distances into $D \in [0, 255]$. Some experimentation was done with different constants, however these were found to work well for the experimental test sets given the centroid scaling used.

2.3. Hardware

The K-means hardware implementation was designed to extend a content classification system [4]. This classification system uses a number of methods from Latent Semantic Analysis (LSA) to map text into a reduced feature space. The mapping of words to features in the 4000 dimensions is done via a mechanism called the Word Mapping Table (WMT). Words are changed into 20-bit memory locations via a hash. For example:

HASH(“MADRID”) = 0x2c563 (101,603)

Output of the hash represents an index in 1MB of SRAM. Values stored in the memory locations are indexes in the 4000 dimension feature vector. When retrieved, these values enable an increment of a counter for the specific feature. The counter bins of the feature vector saturate at 15.

WMTs are produced from different LSA algorithms [5]. The goal of the algorithms is to produce a feature space that promotes uniqueness for concepts. Each method maps words to individual bins. Words such as *CAT* and *CATS* could both point to the same bin in the 4000 dimension if the algorithm determined that grouping the similar words was effective. The traditional method of stemming would definitely group the two words together while Information Theory might find a different information contribution between the two words and thus might separate the two into different bins. The document vectors from the content classification system are well suited for an unsupervised learning environment.

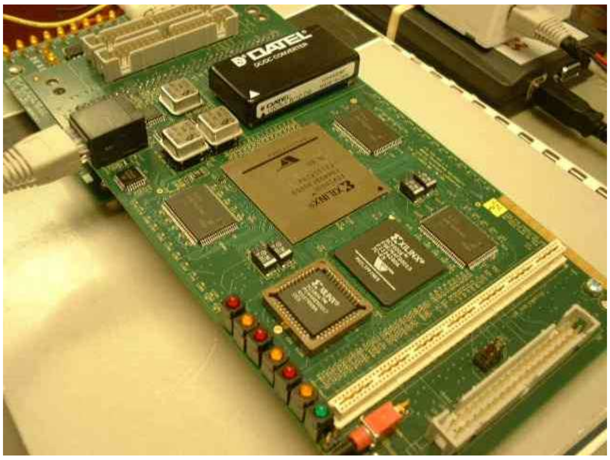

Fig. 2. The FPX platform used to implement K-means Clustering

3. IMPLEMENTATION

3.1. Design Platform

To demonstrate clustering in hardware we synthesized a circuit using VHDL-specified modules. These modules were then used to implement logic for the VirtexE FPGA on the Field Programmable Port Extender (FPX). The FPX platform is an open hardware platform that allows hardware designers to rapidly prototype circuits using VHDL modules [6].

The FPX platform can read documents sent to it directly over the Internet or an Intranet [7]. Wrapper modules separate the application data from the network protocol [8]. Data is sent to circuits in flows carried as a sequence of TCP/IP packets. Modules on the FPX platform receive data through TCP/IP packets [9]. For the K-means clustering hardware, both document vectors and the initial concept vectors (cluster centroids) can be loaded into the FPGA by using UDP/IP packets.

The FPX platform when equipped with two banks of 512MB SDRAM, can cluster 512,000 items (2^{17}). Each record of 2048 bytes (2^{11}) represents an item consisting of the 4000 element 4-bit vector and metadata about the flow. Identifiers and static information, such as the sum of squares of the feature vector, are put into the last 48 bytes.

3.2. Hardware/Software Communications

In order to communicate with the hardware implementation running in an FPX, an interface program was created. This program is used to send data to the hardware and issue a command to start clustering. This program reads the 4000 element document vectors and randomly assigns the documents to initial clusters. The sum of squares for the document vectors are then computed. The random document

[1] The bottom division can be implemented as a bit shift instead of a divide.

assignments are used to calculate the starting concept vectors, the sum of squares and the number of documents contained in each cluster. The document vectors and there cluster assignments are then packed into UDP packets and sent to hardware. The initial concept vector data is packed into UDP packets and sent to the hardware. The document vectors are next loaded into hardware, then the command to start clustering is sent.

After the clustering is started, the results of the clustering are transferred from hardware to software. A program was created that receives the document assignments from each epoch that are sent from hardware. The output at each epoch notifies changes in document assignments. When the algorithm converges or passes the epoch limit, the final document assignments are sent. This information is then formatted into XML files and stored on a network-attached PC.

3.3. Hardware Architecture

The K-means clustering algorithm is comprised of three main operations. These operations are: (1) calculating the distances, (2) identifying the correct cluster assignments, and (3) updating the clusters. In order to implement the algorithm in hardware, three main modules were created: Cosine Distance, Greedy Accept, and Update. In addition to these modules, control modules were needed to load and run the hardware clustering system. A diagram of the K-means clustering hardware is shown in Figure 3. Document vectors are stored in off-chip SDRAM. The 28-bit concept vectors are stored in off-chip SRAM. The reduced 8-bit concept vectors are stored on the FPGA using on-chip BlockRAMs.

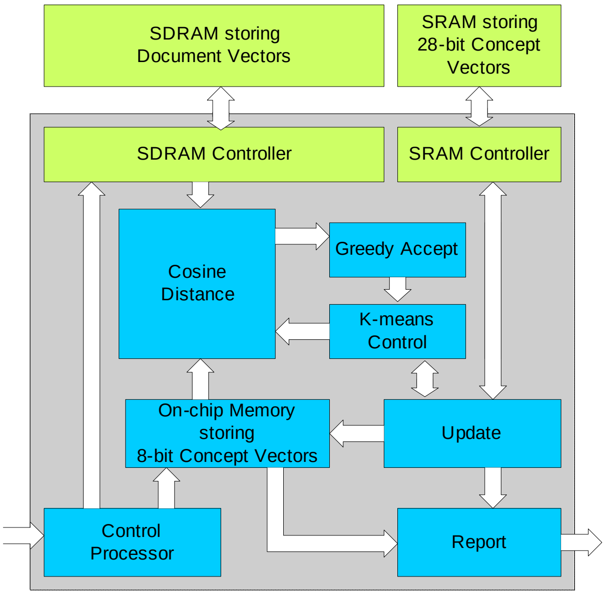

Fig. 3. Hardware Clustering Block Diagram

3.3.1. *Cosine Distance Module*

The cosine distance module is replicated for each cluster that is stored on the FPGA. This allows all the cosine distances of one document vector to each centroid in the FPGA to be calculated in parallel. This module contains all the circuitry required to calculate a cosine distance. This circuitry consists of two square roots, dot product, and a division. This circuitry can produce a cosine distance every 294 cycles (3.675 μs when running at 80 Mhz). The square root function was created using the Xilinx core generator.

In addition to the distance metric circuitry the module also stores all the information related to the concept vector. The concept vectors are stored in on-chip memory on the FPGA. The sum of squares for the concept vector and the number of documents in the cluster are also stored within the module. These values are all used in the calculation of the cosine distance and are also available for other modules such as the update module.

3.3.2. *Greedy Accept*

After calculating the cosine distances (in parallel) a decision is made to determine if the document vector should be assigned to a different cluster. Since K-means is a greedy algorithm the best distance is chosen as the new document assignment. This module compares all the cosine distances that are calculated and chooses the best assignment. If the best distance (the closest one to the true cosine theta calculation) is held by two or more concept vectors the concept vector with the smallest index value is chosen.

3.3.3. *Update*

As the document vector is streamed into the cosine distance module, the update module caches the entire vector. While caching the document vector the update module looks at each 4-bit element and creates a zero-flag indicating whether or not the element is zero. This flag allows the update module to identify only the non-zero document elements and update only the corresponding concept elements. This is useful when the document vectors are sparse.

The module uses the extended 28-bit concept vector values stored in SRAM to recalculate both the 28-bit and the truncated 8-bit concept vectors. The extended vector values are necessary in the update procedure to avoid a cascading precision error. If the truncated 8-bit values were used, the process of expanding the 8-bit value into a 28-bit value would produce a value that would be significantly different from the true 28-bit value. This different 28-bit value would not allow the K-means algorithm to converge.

3.3.4. *Load Processor*

The load processor receives the data from the UDP packets that are sent to the hardware. It then identifies whether the data within the packets represents a document vector, 8-bit concept vector, 28-bit concept vector or a control packet. The module then loads all the document vectors into the FPX's SDRAM. The 28-bit concept vectors are loaded into the SRAM banks. The 8-bit concept vectors are sent to the relevant cosine distance module for storage. When the control packet is sent the load processor starts the clustering by signaling the K-means controller.

3.3.5. *K-means Control*

The heart of the K-means algorithm is contained in the K-means controller. This controller handles all the accesses to document vectors. It starts the cosine distance calculations and depending on information it receives from the greedy accept module it reads another document vector or starts the update procedure for the current document. This controller is designed to output the document assignments after every epoch. It also maintains all the necessary information to determine if the clustering has reached convergence.

3.3.6. *Report*

The report module is use to send information from the clustering hardware to a computer for analysis. This module buffers the document identifiers and the assignments for each of the documents and sends the information out of the hardware. The module is controlled by the K-means controller.

4. RESULTS

4.1. Software Simulation and Performance

Software programs were created to simulate multiple variations on the K-means algorithm. These variations were necessary to test convergence properties and accuracy. Changes included several reduced precision representations, square root versions, and different variations on the cosine distance metric. Most importantly these programs allowed for comparisons between double and integer representations. Based on the simulation results, a hardware implementation was created that utilized 8-bit resolution for the concept vector elements and 4-bit resolution for the document vector elements.

Performance of the K-means software implementations and the hardware implementation can be seen in Figure 4. This shows the average time required for the software to calculate the distances from one document to all concept vectors. As the number of concept vectors increases the average time required to calculate the distances increases. This is due to the fact that software is written to run sequentially.

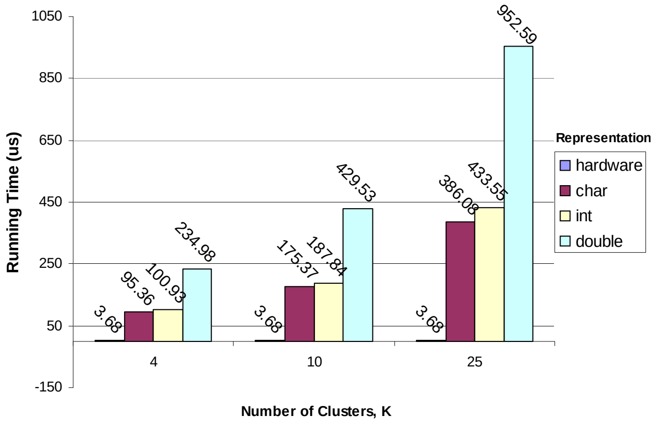

Fig. 4. Average time for each software implementation and the hardware implementation comparing one document to all *K* concepts.

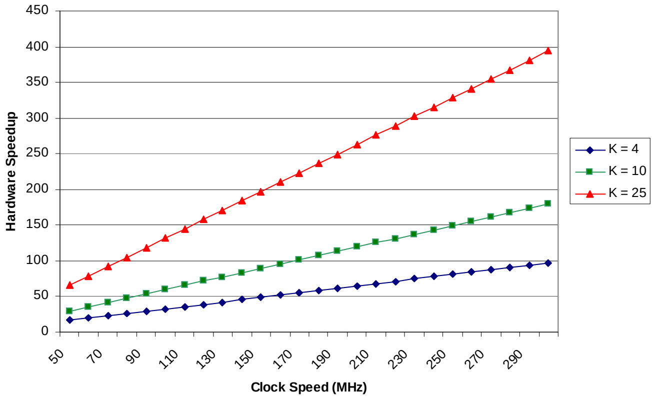

Fig. 5. Hardware speedup in comparison to the character implementation in software as clock frequency increases

4.2. Hardware Implementation Results

The hardware is able to calculate all the distances from a document vector in 3.675 μs when running at 80 MHz. As the clock speed of the circuit is increased the amount of time required to calculate all the distances decreases. The hardware clustering system running at 250 Mhz would require 1.176 μs to produce all the cosine distances. Since all the distances are calculated simultaneously, the timing would not change when the number of concept vectors increase. Given a hardware implementation that runs in 95.36 μs with four clusters and clustering hardware running at 80 MHz, the hardware is 25.95 times faster. As more clusters are added and the hardware clock speed is increased the speed gain of the hardware increases (Figure 5). When the clustering system is implemented on the Virtex4 LX200 FPGA

Resources	XCV2000E Utilization	Utilization Percentage	XC4VLX200 Utilization	Utilization Percentage
Slices	17654 / 19200	91%	19674 / 89088	22%
4-input LUTS	16434 / 38400	42%	19355 / 178176	10%
Flip Flops	29685 / 38400	77%	30048 / 178176	16%
Block RAMs	65 / 160	40%	50 / 336	14%

Table 1. Device utilization for Hardware K-means with four concepts across different platforms

using a clock frequency of 250 MHz, the hardware is 329 times faster then software.

The current hardware implementation is highly parallel. The cosine distance metric is replicated with every concept vector stored within on-chip memory. This means the number of clusters is limited by the on-chip memory and the resource utilization of the distance metric. The Xilinx XCV2000E can support up to fifteen concepts in the on-chip memory. The resource constraint of the clustering hardware for this FPGA are the logic slices. The current system can only contain four distance metrics per VirtexE 2000.

The same hardware design can support up to 25 when implemented on a Xilinx Virtex4 LX200. This Xilinx FPGA has an increased number of logic slices in addition to an increased number (and size) of on-chip memory. The clustering hardware will also experience an increase in the clock frequency when implemented in this FPGA. Table 1 shows the amount of resources utilized when implementing the circuit for four concepts. These constraints allow the Xilinx VirtexE 2000 to support a maximum of four concepts and the Xilinx Virtex4 LX200 to support a maximum of 25.

4.3. Analysis

Utilizing the software implementations, experiments were run on a subset of the CMU 20 Newsgroup Corpus [10]. The CMU corpus is a group of newsgroup postings from 20 different news groups. The subset was determined by stripping the first 2 lines of each file which are the “From:” and “Subject:” lines. Next, the files with less than 100 words were removed. The number of concepts, K was set to 4, 10, and 25. One hundred cluster seeds were generated for each size. Table 2 shows the degree to which each algorithm variation changes the output cluster from a common cluster seed. This allows the comparison of current hardware and future hardware against software implementations running on an Intel 3.60 GHz PC. The software implementations included a double, integer and char representations of concept vectors. Both the 8-bit character and the 16-bit integer versions were tested using both versions of the cosine-theta distance metric. The first distance metric tested is shown in Equation 3. The second is shown in Equation 4. The different versions had no effect on the double representations due to increased precision. The 8-bit character representation using Equation 4 is a simulation of the actual hardware.

	Type &		VI Distance	
K	Cosine Version	Bits	Average	Std. Deviation
4	char(0)	8	3.016	0.0914
	char(1)	8	2.991	0.0895
	integer(0)	16	3.016	0.0914
	integer(1)	16	2.991	0.0895
	double	32	2.940	0.0675
	start seed	-	4.363	0.0008
10	char(0)	8	2.957	0.1467
	char(1)	8	2.974	0.0881
	integer(0)	16	2.984	0.0894
	integer(1)	16	2.974	0.0881
	double	32	2.851	0.0792
	start seed	-	5.270	0.0015
25	char(0)	8	3.579	0.1007
	char(1)	8	3.573	0.0992
	integer(0)	16	3.579	0.1010
	integer(1)	16	3.574	0.0976
	double	32	3.514	0.1005
	start seed	-	6.164	0.0022

Table 2. Variation of Information distance metric showing distance to ground truth for K-Means with varying precision and cosine distance metrics

To compare the different clustering algorithms, Meilă’s Variation of Information [11] metric was used to measure the distance between two clusterings. This is a log scale metric intended to approximate the distance between two clusters. The approximation is of the distance across the lattice of possible clusterings for a set of data points. Lower values indicate closer clusterings. In this case the distance is between a clustering and the ground truth clustering of the CMU 20 newsgroups. Seed denotes statistics for the initial random clusters. These seeds were then run through each of the K-means versions. They are provided to show a baseline of how much each clustering improved, and to give an idea of scale for interpreting these numbers.

Note that in Table 2, for low values of K, such as 4, resulted in the same set of clusterings as the 8-bit character representation. This occurred despite the slight extra precision in the integer version. The additional precision double representations give always result in a clustering that is closer to ground truth. From this table it is safe to conclude that while there is some difference between the double representation and the hardware version of the algorithm, it is a relatively minor difference. It is particularly minor in this application, given the speed gain from hardware as shown in Section 4.1.

5. CONCLUSION

A high speed, parallel K-means algorithm has been synthesised for both a Xilinx Virtex4 LX200 and a Xilinx VirtexE 2000. The circuit was implemented with the Xilinx VirtexE 2000 on the FPX platform. Document vectors with 4000-dimensions and 4-bit precision were clustered with centroids at 8-bit precision. The hardware circuit implemented on the Xilinx VirtexE 2000 clustered four concepts at once. The same implementation on a Xilinx Virtev4 LX200 can cluster 25 concepts at once. The implementation uses more hardware to achieve higher rates of clustering than can be obtained using software algorithms. By scaling values to control numeric precision, we show that the system can find a clustering comparable to a clustering found when using a full floating point software implementation. We also show that by utilizing parallel hardware, the speed of hardware clustering is substantially greater than software. This design of clustering in hardware maintains its speed gains as the number of concepts increases. Thus, it has been shown that software implementations running on an Intel 3.60 GHz XEON PC is outperformed by a fully pipelined architecture running on a Xilinx XCV2000E-8 FPGA (with a clock frequency of 80 Mhz) at a factor of twenty-six.

6. FUTURE WORK

K-means will converge to a local maximum, however it is not guaranteed to converge to a global maximum. Through modifications of the K-means algorithm, such as the use of simulated annealing, it is possible to increase the chances for the algorithm to find the global maximum. Simulated annealing gives the algorithm the chance to jump out of local maximum by probabilistically accepting poor document assignments in the early stages of the algorithm. In later stages the chances of these jumps are reduced, allowing the algorithm to converge to a better local maximum.

7. REFERENCES

[1] *Pattern Classification and Scene Analysis*. John Wiley and Sons, Mar. 1973.

[2] M. Estlick, M. Leeser, J. Theiler, and J. J. Szymanski, “Algorithmic transformations in the implementation of k- means clustering on reconfigurable hardware,” in *FPGA*, 2001, pp. 103–110. [Online]. Available: citeseer.ist.psu.edu/estlick01algorithmic.html

[3] M. Leeser, J. Theiler, M. Estlick, N. Kitaryeva, and J. Szymanski, “Effect of data truncation in an implementation of pixel clustering on a custom computing machine,” 2000. [Online]. Available: citeseer.ist.psu.edu/leeser00effect.html

[4] J. Lockwood, S. G. Eick, D. J. Weishar, R. Loui, J. Moscola, C. Kastner, A. Levine, and M. Attig, “Transformation algorithms for data streams,” in *IEEE Aerospace Conference*, Big Sky, Montana, Mar. 2005.

[5] “Hardware accelerated algorithms for semantic processing of document streams,” in *IEEE Aerospace Conference*, Big Sky, Montana, Mar. 2007.

[6] J. W. Lockwood, “An open platform for development of network processing modules in reprogrammable hardware,” in *IEC DesignCon’01*, Santa Clara, CA, Jan. 2001, pp. WB–19.

[7] J. Lockwood, J. Turner, and D. Taylor, “Field Programmable Port Extender (FPX) for Distributed Routing and Queuing,” in *ACM International Symposium on Field Programmable Gate Arrays (FPGA)*, Monterey, CA, Feb. 2000, pp. 137–144.

[8] F. Braun, J. Lockwood, and M. Waldvogel, “Layered Protocol Wrappers for Internet Packet Processing in Reconfigurable Hardware,” *IEEE Micro*, vol. Volume 22, no. Number 3, pp. 66–74, Feb. 2002.

[9] D. Schuehler and J. Lockwood, “A Modular System for FPGA-based TCP Flow Processing in High-Speed Networks,” in *14th International Conference on Field Programmable Logic and Applications (FPL)*, Antwerp, Belgium, Aug. 2004, pp. 301–310.

[10] (2005) 20 newsgroups. [Online]. Available: http://people.csail.mit.edu/jrennie/20Newsgroups/

[11] M. Meilă, “Comparing clustering - an axiomatic view,” in *Proceedings of the 22nd International Conference on Machine Learning, Bonn, Germany*, 2005. [Online]. Available: www.stat.washington.edu/mmp/Papers/icml05-compare-axiom.pdf