2007-3

Architecture for Document Clustering in Reconfigurable Hardware, Master's Thesis, December 2006

Authors: G. Adam Covington

December 12, 2006

Corresponding Author: gac1@cec.wustl.edu

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 rapidly clusters a half million documents. Documents and concepts are represented as vectors with 4000 dimensions. The circuit was implemented in Field Programmable Gate Array (FPGA) logic and uses four parallel cosine distance metrics to cluster document vectors together.

An exploration of the effect of the integer approximation of the cosine theta distance metric was investigated. Through experiments, measurements were performed to determine the effect of utilizing different numeric representations for the concept vectors. As compared to a full K-means implementation in software, it was found that using carefully chosen integer representations yielded clustering results that were nearly identical to results obtained using full foating-point representations.

Hardware was synthesized and run on the Field Programmable Port Extender (FPX) platform. This implementation on the Virtex-E 2000 FPGA ran 26 times faster than an algorithmically equivalent software running on an Intel 3.60 GHz Xeon. The same architecture was scaled to implement a faster and larger design for the Xilinx-4 LX200. This larger implementation outperformed the equivalent software version on the same Xeon by a factor of 328.

WASHINGTON UNIVERSITY

THE HENRY EDWIN SEVER GRADUATE SCHOOL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ARCHITECTURE FOR DOCUMENT CLUSTERING IN RECONFIGURABLE HARDWARE

by

Gerald Adam Covington, B.S. Computer Engineering

Prepared under the direction of Professor John W. Lockwood

A thesis presented to the Henry Edwin Sever Graduate School of
Washington University in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

December 2006

Saint Louis, Missouri

WASHINGTON UNIVERSITY
THE HENRY EDWIN SEVER GRADUATE SCHOOL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ABSTRACT

ARCHITECTURE FOR DOCUMENT CLUSTERING IN RECONFIGURABLE HARDWARE

by

Gerald Adam Covington

ADVISOR: Professor John W. Lockwood

December 2006

Saint Louis, Missouri

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 rapidly clusters a half million documents. Documents and concepts are represented as vectors with 4000 dimensions. The circuit was implemented in Field Programmable Gate Array (FPGA) logic and uses four parallel cosine distance metrics to cluster document vectors together.

An exploration of the effect of the integer approximation of the cosine theta distance metric was investigated. Through experiments, measurements were performed to determine the effect of utilizing different numeric representations for the concept vectors. As compared to a full K-means implementation in software, it was found that using carefully chosen integer representations yielded clustering results that were nearly identical to results obtained using full floating-point representations.

Hardware was synthesized and run on the Field Programmable Port Extender (FPX) platform. This implementation on the Virtex-E 2000 FPGA ran 26 times faster than an algorithmically equivalent software running on an Intel 3.60 GHz Xeon. The same architecture was scaled to implement a faster and larger design for the Xilinx-4 LX200. This larger implementation outperformed the equivalent software version on the same Xeon by a factor of 328.

I dedicate this work to my parents,
who have remained supportive through the years.
Thank you.

Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank Dr. Lockwood for his guidance and support through the graduate process. Doyle Weishar, John Byrnes, and Alan Ratner provided guidance in the realm of clustering. Their questions and feedback provided the necessary motivation for the work.

Andrew Levine and Charles "Chip" Kastner have been very supportive. No matter how simple the question was, they had a helpful response. Charles Comstock was very helpful in the early stages of software testing. His implementations of cluster comparisons were very important to this work. I would also like to thank the members of the Reconfigurable Network Group in the Applied Research Laboratory.

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.

Gerald Adam Covington

Washington University in Saint Louis
December 2006

Chapter 1

Introduction

With the world's information sources becoming better connected, the amount of information available to be analyzed is ever increasing. This increase is partly due to the Internet, where news sources and market analysts are struggling to parse, categorize, and group the volumes of incoming information.

1.1 Motivation for Clustering

Computers help analysts process data. Automated Processing can be used to categorize and group data. Automatic techniques have been developed in the data mining and artificial intelligence communities that cluster similar data together. The ultimate goal would be to build systems that fully automate the process of organizing data. The operation seeks to minimize the similarity between different groups, or clusters, and maximize the similarity within clusters.

Most of the clustering algorithms are run on general purpose computers. Even though the algorithms have many highly parallel operations, most systems do not take advantage of the parallelism. When Clustering programs are run on a Personal Computer (PC), they perform a long set of sequential operations. Much of the computation could be performed in parallel.

Parallel hardware circuits can be built to perform higher levels of parallel computation than would be possible on general purpose processors. Application Specific Integrated Circuits (ASICs) could be fabricated to perform high speed data clustering operations. But, there are a few drawbacks to building ASICs: the high cost of fabricating an ASIC to solve a specific computational problem is unattractive; further, ASICs are static circuits and do not provide flexibility to change or reprogram.

An alternative to an ASIC design is to use a Field Programmable Gate Array (FPGA). FPGAs are programmable devices and can be reconfigured in the field to perform different computations. Reconfigurablity enables new circuits to be implemented even as parameters of the algorithm are modified. The FPGA provides a similar type of functionality as that of an ASIC, but without high Non-Recurring Engineering (NRE) cost and at only a slightly lower level of performance.

1.2 Objectives of Thesis

Hardware-accelerated clustering systems are needed so that systems cluster data scale in performance with the rate of increase in raw data generated throughout the world. Today's clustering algorithms, which are implemented on general purpose CPUs, require large amounts of time to cluster. As the amount of raw data increases faster than the rate at which performance scales on the PCs, the performance of software-only based implementations will fall far behind. The purpose of this work is to implement an architecture that dramatically increases the throughput of clustering algorithms.

An architecture was created to implement clustering operations efficiently in hardware. While the implementation is well-suited for FPGA hardware, the ideas apply to other types of devices that perform parallel computation. The main objectives are as follows:

- Identify aspects of the clustering algorithm that can be accelerated by hardware
- Evaluate algorithmic features in software before implementing hardware
- Create a clustering hardware architecture framework
- Build hardware circuits that compute multiple distance metrics, which will be shown to be the most computationally intensive part of clustering algorithms
- Demonstrate how well hardware clustering circuits with different parameters cluster data in different data sets

Chapter 2

Background

The goal of clustering is to group similar content together. Distance metrics define how we measure the similarity of any pairs of data. Clustering algorithms define a similarity metric that determines the distance from a document to a point that represents a cluster. The *intra-cluster distance* measures how far apart pairs of data appear within a cluster. The *inter-cluster distance* measures how far apart pairs of data appear when they are assigned to different clusters. The objective of clustering is to maximize inter-cluster distance and minimize intra-cluster distance.

The best similarity metrics to use depends on the data that is being clustered. There is no single algorithm or similarity metric that clusters all types of data. The algorithms and metrics are tailored to specific data representations. The most widely known clustering algorithm is called K-means. More complex methods, such as Spectral and Co-clustering, have been developed as well.

2.1 Clustering Algorithms

Clustering algorithms can be separated into two categories. These categories define how the clusterings are created. The first type, *divisive*, starts with one large cluster and divides the cluster into smaller groups. The second type, *agglomerative*, starts with each data element as its own cluster and seeks to create clusters by joining smaller clusters together. The algorithms that are described within this work are divisive clustering algorithms.

Studies have shown that when documents are represented with a *bag-of-words*, a divisive clustering performs better on the average [1] than an agglomerative clustering. The explanation for this is that the agglomerative clustering algorithms start with

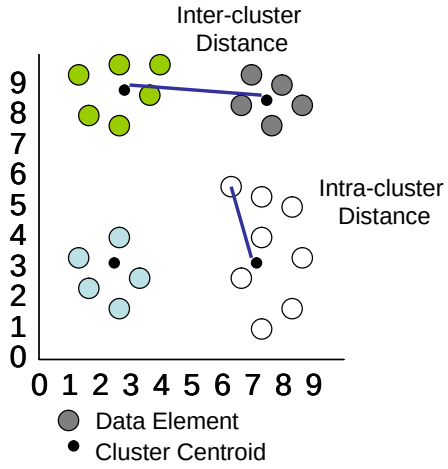


Figure 2.1: Example of four clusters in two dimensions

local neighborhood comparisons, but these local comparisons can allow pairs of data, which are in fact similar, to not be combined together. Divisive algorithms start by considering similarities of all the data elements at once.

2.1.1 K-means

The K-means algorithm implements a divisive clustering and was first discussed by Duda and Hart [2]. The algorithm uses a similarity metric to assign all documents to one of k clusters. The clusters are represented as an average of all documents contained within the cluster. This average can be thought of as the centroid of the cluster.

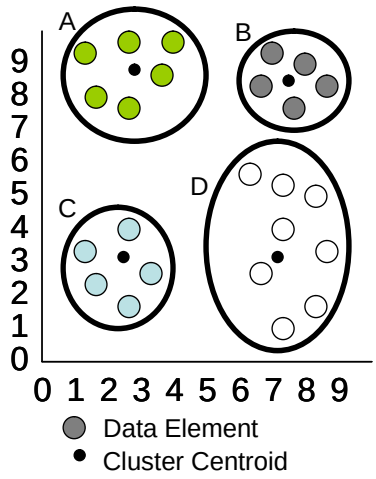


Figure 2.2: Example of K-means output with K equal to four in two dimensions

A simple two dimensional case for K-means clustering is shown in Figure 2.2. The K-means algorithm set with $k = 4$ results in four clusters represented by A, B, C, and D.

The K-means algorithm operates as follows:

1. Assign document vectors, $d_i \in D$, to a cluster using an initial seed.
2. Initialize cluster centroids, C, from initial document assignments.
3. For each document $d \in D$
   (a) Recalculate distances from document d_i to centroids (C_1,C_2,...,C_k), and find the closest centroid C_{min}.
   (b) Move document 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.

The initial seed clusters can be either assigned or generated by randomly assigning documents to clusters.

K-means has been used in the clustering of images. A hardware implementation of K-means to cluster hyperspectral images was created by Estlik et. al. [3] and Leeser et. al. [4]. Hyperspectral images have 224 16-bit channels which can be thought of as features to cluster per pixel [25].

Centroid Update

Cluster centroids are initialized by averaging document vectors across all members in each cluster. We recalculate the centroids when documents are added or removed from a centroid. We add and 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.

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

2.1.2 Spectral Clustering

Spectral clustering allows for efficient clustering of high dimensional sparse data sets. Spectral algorithms identify the top features or elements within document space and map all documents into a new space that contains these top features. This dimensionality reduction helps improve the throughput and speed of clustering.

The main Spectral Clustering algorithm is defined as follows:

Given a document $d \in D$,

1. Create affinity matrix A defined by
   $A_{ij} = \exp(-||(d_i - d_j)^2||/2\sigma^2)$
   $\forall\ i \neq j$ and $A_{ii} = 0$
2. Construct the matrix $L = N^{-1/2}AN^{-1/2}$,
   where N is the diagonal matrix whose (i, i) element is the sum of A's i-th row
3. Find k largest eigenvectors $(x_1x_2...x_k)$ of L
4. Form matrix $X = [x_1x_2...x_k]$ by stacking eigenvectors in columns
5. Renormalize X's rows to have unit length
6. Treat each row of X as a data element and cluster into k clusters using K-means
7. Assign original data element d_i to cluster to cluster j
   if and only if row i of matrix X was assigned to cluster j

Spectral clustering uses K-means to cluster the matrix of eigenvectors. Implementing K-means in hardware would allow for a high speed spectral clustering algorithm in the future. The only difference is the addition of mapping the data elements to a new space using the top k eigenvectors. For more information in spectral clustering please refer to [5].

2.1.3 Co-Clustering

Information Theoretic Co-Clustering has been discussed and implemented by Rohwer [6] and Dhillion [7]. The Co-Clustering algorithm allows for clustering to occur in two dimensions at once. Given a set of documents represented as feature vectors D, the algorithm can cluster features into l groups and the documents into k groups.

The first step in Co-Clustering is to create a matrix of the document vectors. Each column of the matrix represents a document. Each row of the matrix represents a separate word or feature found in the documents. This matrix is then normalized by its magnitude.

An initial mapping is then used to assign each document to a value in the range of [0,k]. The words or features are also given an initial mapping to range [0,l]. These initial ranges are then used to produce what is known as the cluster matrix. The cluster matrix is a LxK matrix. This matrix is then used to determine a value that represents the amount of information contained within it. The metric generally used is the Mutual Information metric that can be seen in Equation 2.10.

When clustering the documents, the documents are selected one at a time. The document is then randomly assigned to a different value between [0,k]. A new cluster matrix is generated using this new mapping. From this cluster matrix the new MI value is computed. This new value is then compared to the MI value of the old assignment. If the MI is better, the assignment is allowed and clustering continues. Otherwise, the assignment is reverted back to the original and another document is selected.

The clustering of the rows maintains the same procedure as the column clustering. The difference is that the assignments are modified for the range of [0,l]. Clustering of the dimensions is done alternating between each other after cycling through all elements in the current dimension.

The complexity of this algorithm occurs in the normalization of the original raw data matrix and the calculation of the MI distance value.

2.2 Distance Metrics

In order to group data elements that are similar, a measure of similarity must be defined. This measure of similarity can be thought of as the distance between one data element and another. This distance/simiarlity metric can be defined in many ways. Some individuals would use a Euclidean distance to determine similarity. Although this does provide a distance, it does not provide much information about the similarity of the data. Since we not only want a distance between two data elements but also to know the likeness, we need to explore other metrics.

2.2.1 Manhattan Distance

The Manhattan distance, or city block distance, provides the absolute distances between two data elements. Like the Euclidean distance, this metric describes a distance between points in space. In a two dimensional grid the calculation sums the absolute values of the difference between each of the vector elements. Given the example shown in Figure 2.3 the Manhattan distance is nine. The general Manhattan distance equation is as follows:

$$ManhattanDistance = \sum_{i \in 0..N-1} |\vec{A}_i - \vec{B}_i| \tag{2.1}$$

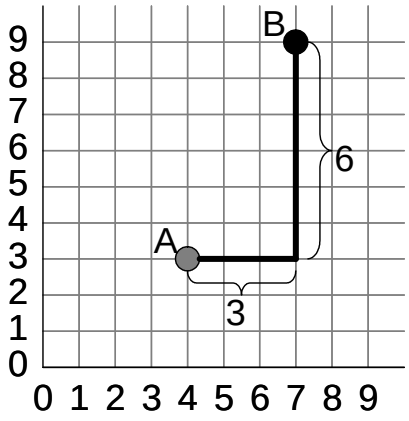


Figure 2.3: Example Manhattan Distance calculation in two dimensions, $|A_x - B_x| + |A_y + B_y| = |4 - 7| + |3 - 9| = 3 + 6 = 9$

2.2.2 Cosine Theta

In most metrics, the size of the vectors being compared affects the calculated distance. The Cosine Theta distance metric allows for two vectors to be compared; however, their relative sizes are not taken into consideration in the calculation. This metric operates by identifying the angle, θ, between the vectors being compared. As can be seen in Figure 2.4, the angle between $\vec{A}$ and $\vec{B}$ can tell us how far apart the vectors occur.

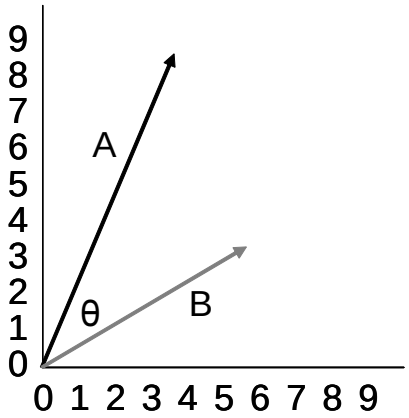


Figure 2.4: Theta represents the angle between $\vec{A}$ and $\vec{B}$

Instead of determining the actual angle, θ, we can determine the numeric value of cosine theta. This value is derived from the Equation 2.2. Given a document vector $\vec{D}$, and a centroid vector $\vec{C}$, the spherical distance between $\vec{D}$ and $\vec{C}$ is defined as:

$$\cos(\theta) = \frac{\vec{D} \cdot \vec{C}}{|\vec{D}| \cdot |\vec{C}|} \tag{2.2}$$

Cosine theta distance D is ranged $D \in [0, 1]$.

2.3 Cluster Comparison Measures

Comparing the results of two clusterings is difficult. Even comparing two clusterings of the same algorithm can be problematic. The main problem is defining what metric to use as a comparison. Methods of comparing clusterings include counting pairs, set matching and variation of information. For each of the methods described in the

sections that follow the clusterings C is comprised of clusters C_1, C_2, ...C_K. The clusterings, C, are a partition of the data set.

2.3.1 Counting Pairs

Clusters can be compared by the method of counting pairs. Counting pairs seek to identify the points that agree and disagree between two clusterings C and C'. The four classes that data elements can fall under follow:

N_{11} : number of data elements that are in the same clusters in both C and C'
N_{00} : number of data elements that are not in the same clusters in both C and C'
N_{10} : number of data elements that are in the same cluster in C but not C'
N_{01} : number of data elements that are in the same cluster in C' but not C

These counts will always sum to $n(n-1)/2$. This becomes important in the metrics that utilize these counts.

Two asymmetric criteria W_I, and W_{II} were proposed. These metrics, first proposed by Wallace [8], are defined as follows:

$$W_I(C, C') = \frac{N_{11}}{\sum_k n_k \frac{(n_k-1)}{2}} \tag{2.3}$$

$$W_{II}(C, C') = \frac{N_{11}}{\sum_{k'} n'_{k'} \frac{(n'_{k'}-1)}{2}} \tag{2.4}$$

They define the probability that a pair of data elements that are the same cluster under C are also in the same cluster under clustering C'.

A symmetric criteria was introduced by Fowlkes and Mallows [9]. The Fowlkes-Mallows criterion is defined below.

$$F(C, C') = \sqrt{W_I(C, C')W_{II}(C, C')} \tag{2.5}$$

This index is used to compare clusterings by first identifying the expected value giving independent clusterings. This expected value then subtracted from the index value given from the clusterings C and C'. This value is then normalized by the range. The resulting value is a normalized index from zero to one. A value of one is the result of identical clusterings. It is possible that the indices produced from the normalization could be negative [10].

An adjusted Rand index, which was introduced by Hurbert and Arabie [11], is similar to the Fowlkes-Mallows criterion.

$$R(C, C') = \frac{N_{11} + N_{00}}{\frac{n(n-1)}{2}} \tag{2.6}$$

Both the Fowlkes-Mallows and the adjusted Rand index require a baseline. This baseline is recomputed for every pair of clusterings. The baseline is derived as an expectation under a null hypothesis. The null hypothesis requires two properties: the first property is that the two clusterings are sampled independently; the second property is that the clusterings are derived from a set partitions where the number of elements in a partition is set [9] [11]. These two properties are not followed when clusterings are generated using K-means. The number of data elements within each cluster is not a parameter that is set at the start of K-means, but instead the number of clusters that are to be found is a parameter.

There are other metrics and comparisons, however the results of these other metrics also do not give a good insight into the effectiveness of the clusterings. The metrics that have been described so far do not take into consideration the amount of information that is held within the clusterings themselves.

2.3.2 Set Matching

Set matching allows for clusterings to be compared without the assumptions of how the clusterings were formed. The metric that was given by Larsen [12], is shown in Equation 2.7. This metric is asymmetric and returns a one when the clusterings are the same.

$$L(C, C') = \frac{1}{K} \sum_{k} \frac{2m_{kk'}}{n_K + n'_k} \tag{2.7}$$

Another asymmetric metric is defined by Meilă and Heckerman [13]. This metric also returns a one when the clusterings are the same. The metric operates by identifying the best match for each cluster in C to a cluster in C'. After a minimum of K or K' matches are made, the index is determined by Equation 2.8.

$$H(C, C') = \frac{1}{n} \sum_{k'=match(k)} m_{kk'} \tag{2.8}$$

Each of the metrics must identify the best match for each cluster in clustering C to a cluster of C'. After a match has been made, the contributions of each data element that is contained within both clusterings is then calculated. These metrics ignore the data elements that are not found in both of the matched clusters. This is depicted very will by Meilă in [10].

The main problem with the above metrics involves how they account for set matching. Neither metrics are well-defined when there is an unmatched cluster. This problem is difficult to solve. Further, asymmetric metrics are not easy to interpret.

2.3.3 Variation of Information

The Variation of Information (VI) provides a better cluster comparison criteria. Unlike set matching metrics, this metric is not concerned with the relationships between pairs of data elements. The metric can be broken down into two steps: the first involves finding the entropies and the second involves computing the Mutual Information (MI) between clusterings.

The first step computes the entropies of each of the clusterings. This is determined using Equation 2.9.

$$H(C) = -\sum_{k=1}^{K} P(k) \log P(k) \tag{2.9}$$

The second step, determining the MI, provides an understanding of how much information one clustering has about another. This calculation when applied to clusterings is defined in Equation 2.10.

$$I(C, C') = \sum_{k=1}^{K} \sum_{k'=1}^{K'} P(k, k') \log \frac{P(k, k')}{P(k)P(k')} \tag{2.10}$$

Given the Equation 2.9 and Equation 2.10, the Variation of Information can be defined.

$$VI(C, C') = H(C) + H(C') - 2I(C, C') \tag{2.11}$$

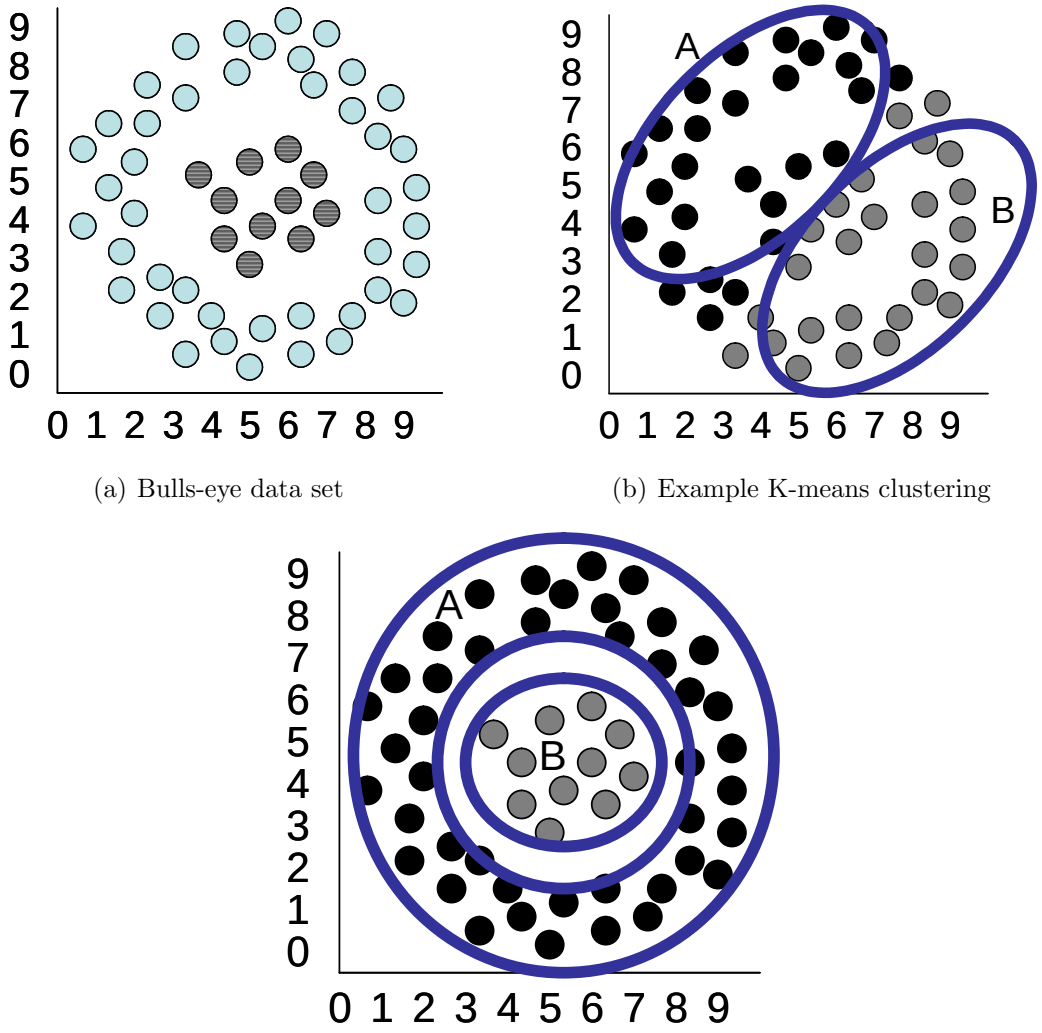
This equation can be represented as the sum of two positive terms. The first term $(H(C) - I(C, C'))$ relates the amount of information is lost in clustering C. The second term, $H(C') - I(C, C')$, relates the amount of information that there is still to gain from clustering C'. The VI distance has an upper bound when comparing two clusterings that contain K clusters each. The upper bound for the VI distance is equal to 2 log K. For more information on the VI metric please refer to [10].

2.4 Challenges

The utility of a clustering algorithm depends on the data that is to be clustered. Certain algorithms are better suited for certain data sets. The bulls-eye data set, shown in Figure 2.5(a), demonstrates the differences between K-means and Spectral clustering. The bulls-eye data set is made up of one central cluster that is spherical. The spherical cluster is then ringed by at least one cluster that is in the shape of a ring.

K-means works well on most data sets but is not at all effective for patterns of clusterings on a bulls-eye data set. Figure 2.5(b) shows that K-means would have to find two centroids that fit the data set. Since the true centroids are located at the same point (i.e. the center of the spherical cluster), K-means will never find two clusterings that are equivalent to a bulls-eye. Spectral clustering, on the other hand, works well on the bulls-eye (Figure 2.5(c)). Currently, there is no single algorithm seems to work well all of the time.

The computation of clustering requires extensive input and output of document vectors to and from memory. For instance, K-means performs an iteration that involves



(a) Bulls-eye data set

(b) Example K-means clustering

(c) Example Spectral clustering

Figure 2.5: Bulls-eye dataset and example clusterings using K-means and Spectral clustering

selecting a data vector from memory (or some sort of temporary storage) and comparing it to all the concept vectors (aka centroids). Each document vector has an assignment to a concept vector. This assignment is updated in each iteration. Architectures need to consider how locality of data can be maintained to limit the I/O bottleneck.

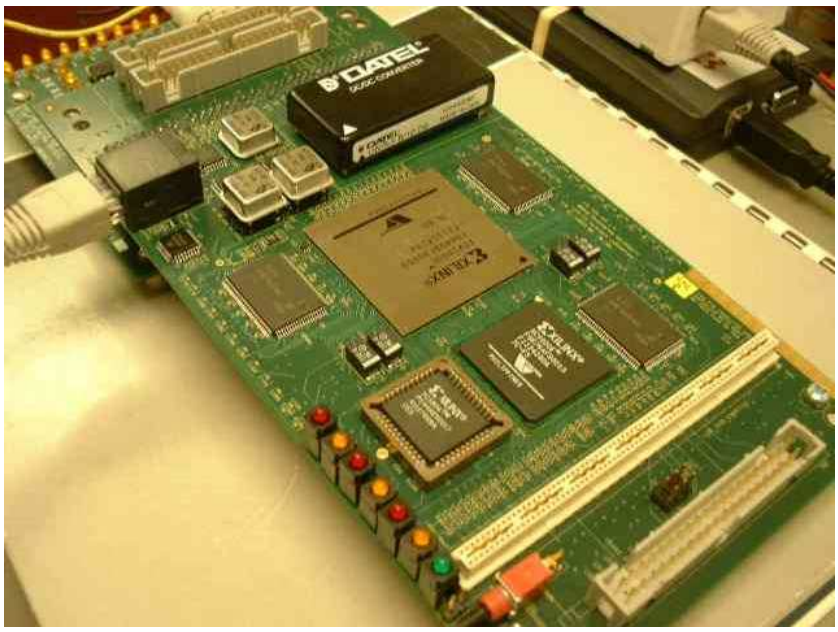


Figure 2.6: The FPX platform used to implement K-means Clustering

2.5 Implementation Medium

To perform clustering in hardware, we synthesized circuits using VHDL-specified modules. These modules were then placed and routed into logic blocks on a VirtexE 2000 FPGA on the Field Programmable Port Extender (FPX) platform (Figure 2.6). The FPX platform is an open hardware platform that allows hardware designers to rapidly prototype circuits using VHDL modules [14]. This platform has been used for numerous circuits, including a language identification circuit called HAIL [15]. The FPX platform contains a Xilinx VirtexE 2000, two banks of SDRAM, and two banks of zero buffer turnaround (ZBT) SRAM. All the applications and infrastructure was implemented on the Xilinx FPGA.

The FPX platform was designed to process data sent to it over a network [16] [17]. Data can be sent to FPX modules using ATM cells or Internet packets [18]. Modules on the FPX platform decode traffic flows sent as a sequence of TCP/IP packets. Wrapper modules separate the application data from the network protocol [**?**]. For the K-means clustering module, both document vectors and the initial concept vectors (cluster centroids) are loaded into the FPGA via packets.

2.6 Documents to Vectors

A content classification system was designed. The system uses a number of methods from Latent Semantic Analysis (LSA) to map text into a reduced feature space [19]. This system, known as the Automated Front End (AFE), maps words to features in 4000 dimensions via a mechanism called the Word Mapping Table (WMT) [19] [20]. Words are changed into 20-bit memory locations via a hash. For example:

$$\text{HASH}(\text{“MADRID”}) = 0\text{x}2\text{c}563\ (101{,}603)$$

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 [20]. 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 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.

Chapter 3

Implementation of Clustering in Hardware

A trace of the K-means algorithm execution, as it was run in software, indicated that the distance calculations consumed most of the CPU's running time. This statistic indicated that when porting the K-means algorithm to hardware, a significant benefit could be achieved if a high level of parallelism could be obtained when computing the distance calculations. Such parallelism would allow hardware to achieve much faster clustering than software. Preliminary results were first published in the International Conference of Field Programable Logic in 2006 [21].

3.1 Hardware Considerations

When targeting an implementation of an algorithm in hardware, it is necessary to identify the the nature of the computation. An FPGA platform can perform floating point arithmetic; however, today's devices are better suited for integer arithmetic. Floating point units implemented in soft logic are not as dense as integer units and could reduce the amount of parallelism available within the hardware. FPGAs can perform floating point arithmetic if absolutely necessary, but if the arithmetic can be mapped into integer arithmetic, it can be implemented far more efficiently.

The use of memory is another major factor that affects how well an algorithm performs in hardware. The Xilinx Virtex FPGA family all have embedded memories on the chip. This allows for FIFOs, buffers, and caches with on clock cycle of latency to be implemented. This delay is significantly smaller than accessing external (off-chip) memories.

3.2 K-means Modifications

Because the most time-consuming aspect of K-means clustering was the distance calculation, the first goal in the implementation of hardware was to parallelize this process. The distance calculation was parallelized by increasing the number of calculations that can be performed at once. The parallelization was achieved by storing k concept vectors in on-chip memory. This allowed the algorithm to compare k centroids to each document that was read from off-chip memory in parallel.

3.2.1 Centroid Scaling

Since the amount of on-chip memory on an FPGA is a fixed, it is important to make good use of this resource. Given that our system has document and centroid vectors with 4000 dimensions, we need to store a 4000-dimensional vector for each concept. Concept vectors (centroids) were chosen to have an eight bit representation, thus the amount of storage needed to represent each concept was 4K-bytes.

The centroid of a concept is defined as the average of the documents contained by that concept. In general, this average is a floating point number. But this value can be scaled into an integer representation, C_{scaled}. The way a concept, C, in this work was scaled to an integer, C_{scaled}, was as follows:

$$\vec{C}_{scaled} = \min\left(255, \frac{2 \cdot \vec{C}_{unscaled}}{\min(C_{count}/16, 1)}\right) \tag{3.1}$$

The minimum function in the denominator was used to avoid a division by zero. Without this function, a division by zero would occur if the document count had a value below sixteen. The minimum function in the outside of the division ensures that the final scaled value fits within the 8 bits allocated per centroid dimension.

3.2.2 Cosine Theta Distance Mapping

The Cosine Distance metric is defined by the equation below. Given a document vector $\vec{D}$, and a centroid vector $\vec{C}$, the cosine theta distance between $\vec{D}$ and $\vec{C}$ is defined as:

$$\cos(\theta) = \frac{\vec{D} \cdot \vec{C}}{|\vec{D}| \cdot |\vec{C}|} \tag{3.2}$$

The range of the cosine theta distance, D, is $D \in [0, 1]$. In order to represent the cosine distance as an integer, it was necessary for the value to be scaled into a larger range suitable for fixed point arithmetic.

In this work, the integer representation was determined using the following equation:

$$\cos(\theta)_i = \frac{16\left(\frac{16(\vec{D} \cdot \vec{C})}{|\vec{C}|}\right)}{|\vec{D}|} \tag{3.3}$$

To further reduce the number of divisions necessary, the lower division can be implemented using a bit shift instead of a divide, which consumes no resources on an FPGA.

$$\cos(\theta)_{i_2} = \frac{16(\vec{D} \cdot \vec{C})}{\min((|\vec{C}| \cdot |\vec{D}|)/16, 1)} \tag{3.4}$$

The constant multiple of 16 was chosen to scale the range of distances into $D \in [0, 255]$. Experimentation was performed with different constants to determine which value of the constant achieved the best performance using the experimental data.

Chapter 4

Algorithmic Performance Analysis

To measure the effect of limited precision on the K-means clustering algorithm, a series of tests were performed. In these tests, a K-means algorithm was implemented in software that allowed the precision of the arithmetic to be treated as a parameter. The experiments determine the effect of precision on the clustering results.

4.1 Experimental Setup

In order to compare two different clusterings, two different metrics are utilized. The first is the Mutual Information (MI) distance. The MI distance determines the amount the clusterings are similar. The second metric used is Meilă's Variation of Information (VI) metric [22]. The VI metric measures the distance between a group of clusterings along the lattice of possible clusterings. A distance of zero for the VI metric indicates that the clusterings being compared are the same.

Four different experiments were performed. The first executes a full floating point K-means clustering. The second uses mapped cosine theta equation shown in Equation 3.4, with a normal concept vector representation. The third experiment implements a normal cosine theta calculation with a concept vector reduced to the hardware mapping of 8-bits. The fourth experiment implements both the cosine theta mapping with the concept vector reduction to 8-bits. Each of the experiments varied the amount of seeding from 0 to 100 percent. A seeding of zero represents a completely random seeding. For each of the seeding percentages, the experiments were run 25 times with a different random seeding for each of the 25. Each of the four experimental setups used the same seeding for the 25 runs.

4.2 Data Sets

Two types of data were used to test the precision and performance of the clustering system. The first data type consisted of synthetic data, defined by four categories. Each of the four categories was given a range of values that represent that concept. As shown in Figure 4.1, the first category is assigned to the first 1000 elements. The second category is assigned the next 1000 elements. There are two data sets that are derived from this synthetic data. The first is comprised of 120 documents, and each concept contains 30 documents. The second set is comprised of 4000 documents, and each concept containing 1000 documents.

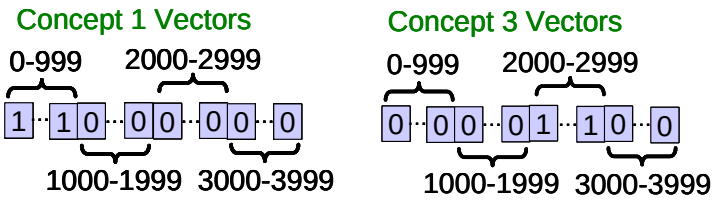


Figure 4.1: Each concept is given a separate range set to 1's

The second set of data was created from the CMU20 newsgroups [23]. To create document vectors, documents in the newsgroups where passed through the AFE system. The AFE system uses the Word Mapping Table (WMT) that has been loaded into it to map the words within a document to a feature vector of 4000. The WMT used in the creation of the document vectors was varied by using two different tables. The first table was created using the WU method, while the second was created using the HNC / Fair Isaac method. Both methods are discussed in [20]. The frequency of each feature is then calculated. This frequency vector, also known as the count vector, is then used as the document vector that will be clustered.

Each of these WMT methods was tested using two different techniques. The first technique used the WMT generation methods normally. This meant that the words within the training documents were mapped into a total of 4000 dimensions. The second method scaled the learned word mappings from the training documents to the first 2000 dimensions. All words that were not found in the training documents are randomly mapped to the last 2000 dimensions. These WMT's were referenced for the duration using the following mappings:

- WU method normal : WUSTL

- WU method limiting training to 2k dimensions : WUSTL 2k
- HNC / Fair Isaac normal : HNC
- HNC / Fair Isaac limiting training to 2k dimensions : HNC 2k

The CMU20 newsgroups was then separated into two smaller subsets. Each of the subsets contained four concepts. The differences between the sets includes the number of documents per concept and the concepts chosen for the set. Table 4.1 and Table 4.2 show the concepts and number of documents for the CMU20 subsets.

Category	**Number of Documents**
rec.autos	171
rec.sport.baseball	163
sci.crypt	197
talk.politics.mideast	621

Table 4.1: CMU20 subset 1 categories and the number of documents per category

Category	**Number of Documents**
rec.sport.baseball	652
comp.sys.ibm.pc.hardware	620
sci.med	702
alt.atheism	650

Table 4.2: CMU20 subset 2 categories and the number of documents per category

4.3 Synthetic Data Experiments

The synthetic data set was created so that each category would be guaranteed to be orthogonal. This data set also ensured that the true category assignment was known. This true category assignment, or ground truth, was used to demonstrate K-means clustering. All four experiments were run using this data set, but since the clustering results were very similar only output from one experiment will be presented.

Figure 4.2 shows the VI distance of the hardware simulation to the full K-means implementation. When the initial assignments are seeded by 60% or greater, the hardware implementation maintains the correct clustering (VI distance of zero). The VI distance is always below a VI distance of .01. Since the maximum VI distance available for the two clusterings containing four clusters is 1.2, the clustering does

perform well. The reason the result clusterings have a variation of the VI distance is that occasionally two of the categories are combined into one by the algorithm.

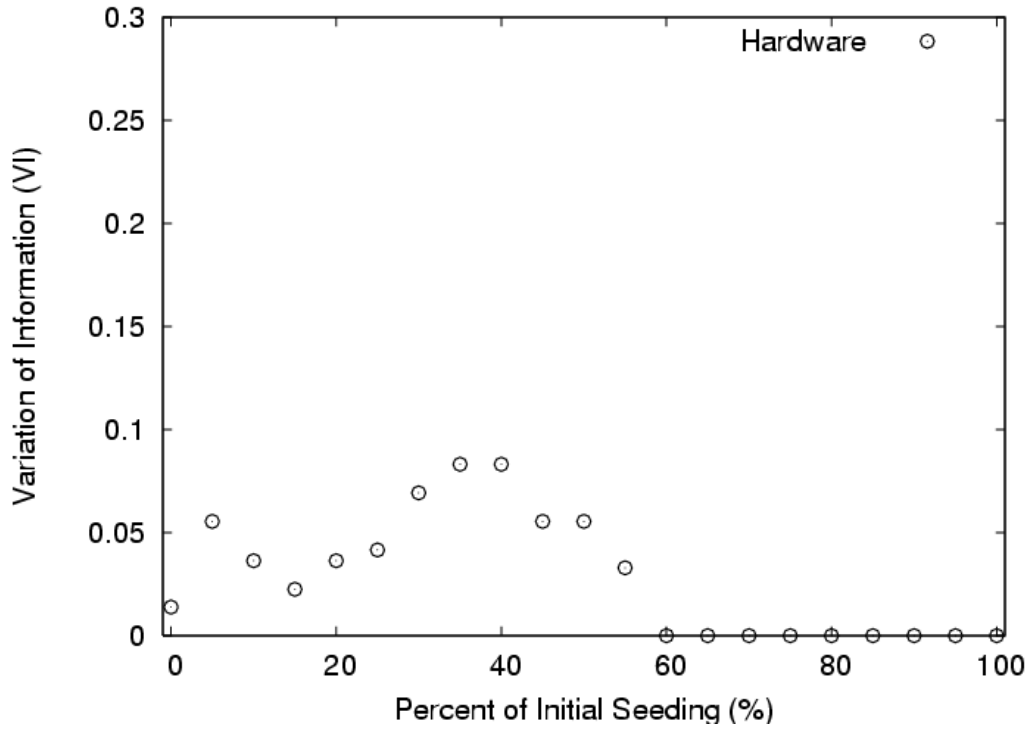


Figure 4.2: Average VI distance of Hardware Simulation with Synthetic Data (120 data vectors)

Another way to look at the output of clusterings is to provide a confusion matrix. As seen in Table 4.3, the columns are true labels of the documents. The rows are the output of a clustering algorithm. The central idea of a confusion matrix is to quickly identify the clustering results. An ideal cluster would have each of the column values located in exactly one row. When the initial assignments were seeded 60% or greater, the confusion matrix shown in Table 4.3 was created. Since confusion matrices show the results of only one clustering experiment, the rest of the sections will rely on the VI distance for comparisons.

	Cat 0	**Cat 1**	**Cat 2**	**Cat 3**
Cluster 0	30	0	0	0
Cluster 1	0	30	0	0
Cluster 2	0	0	30	0
Cluster 3	0	0	0	30

Table 4.3: Confusion Matrix for a Synthetic Experiment with 120 data vectors

4.4 Cosine Distance Mapping

Using the cosine distance metric discussed in Section 3.2.2 caused the algorithm perform differently than a full floating point version. This section explores how the cosine theta approximation affects the mapping on the output clusterings. In addition, the effect of using the HNC WMT, WUSTL WMT and their variations (normal training and 2k mapped) was explored.

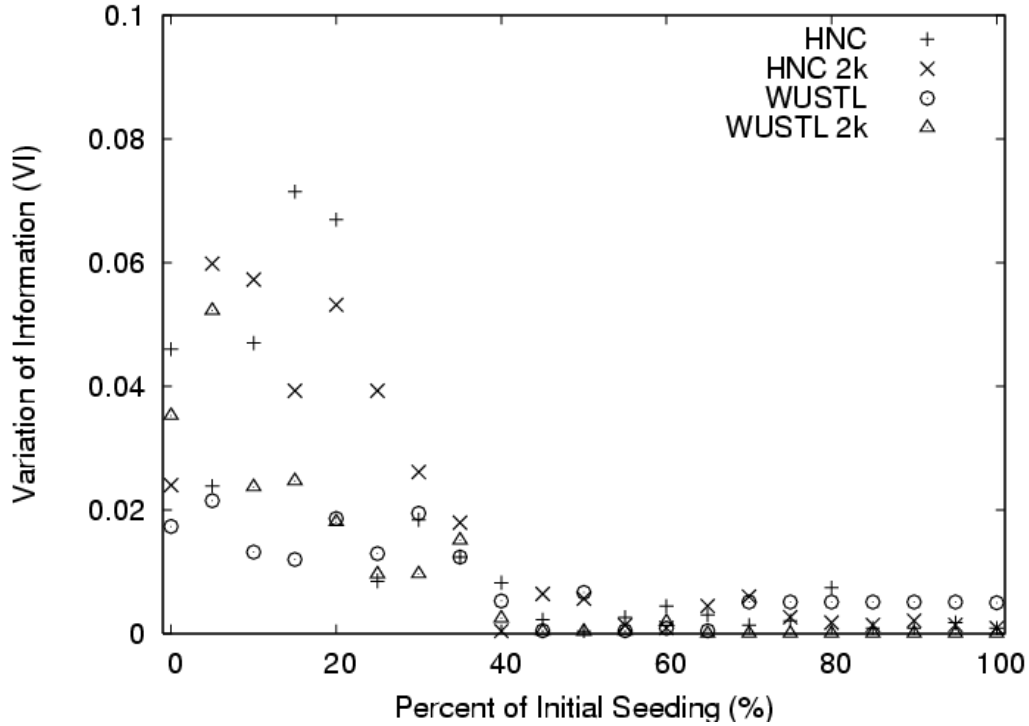


Figure 4.3: Average VI distance of mapped cosine theta to full floating point K-means with same Seeding using the CMU20 subset one

Figure 4.3 shows the average VI distance of the Cosine Mapped clusterings to the full floating point clusterings on the CMU20 subset one. The graph shows the two different word mapping tables with the two training methods discussed previously. The VI values for the Cosine Mapping are very close to zero even with a completely random initial seeding. When the seeding is greater than or equal to 40%, the average VI distance for Cosine Mapping is essentaily zero. A value of zero for the VI distance means that the clusterings are identical. So, a value that is close to zero shows that the mapping has very little effect on the overall clustering algorithm. It can be seen that the WUSTL WMT provides a lower VI distance when the seeding is less than 40 percent. When the seeding is greater, all the WMTs are very close to a VI distance of zero; however, the WUSTL 2k WMT does provide a value closer to zero.

4.5 Concept Vector Reduction

Scaling of the concept vectors affects the behavior of the clustering algorithm. An experiment was run to compare the standard floating point implementation to the concept vector scaling version that was discussed in Section 3.2.1. As shown in Figure 4.4, the concept vector reduction does produce an average VI distance that is larger than the the cosine distance mapping. It is important to note that most of the VI distances for the centroid scaling have a value of .1 or less. It was expected that the reduction of the Concept Vector to 8-bits would produce a VI distance greater than one. Given that the upper bound on the VI distance when comparing two clusterings is approximately 1.2, a value closer to 1.2 would show a significant variance in the result clusters.

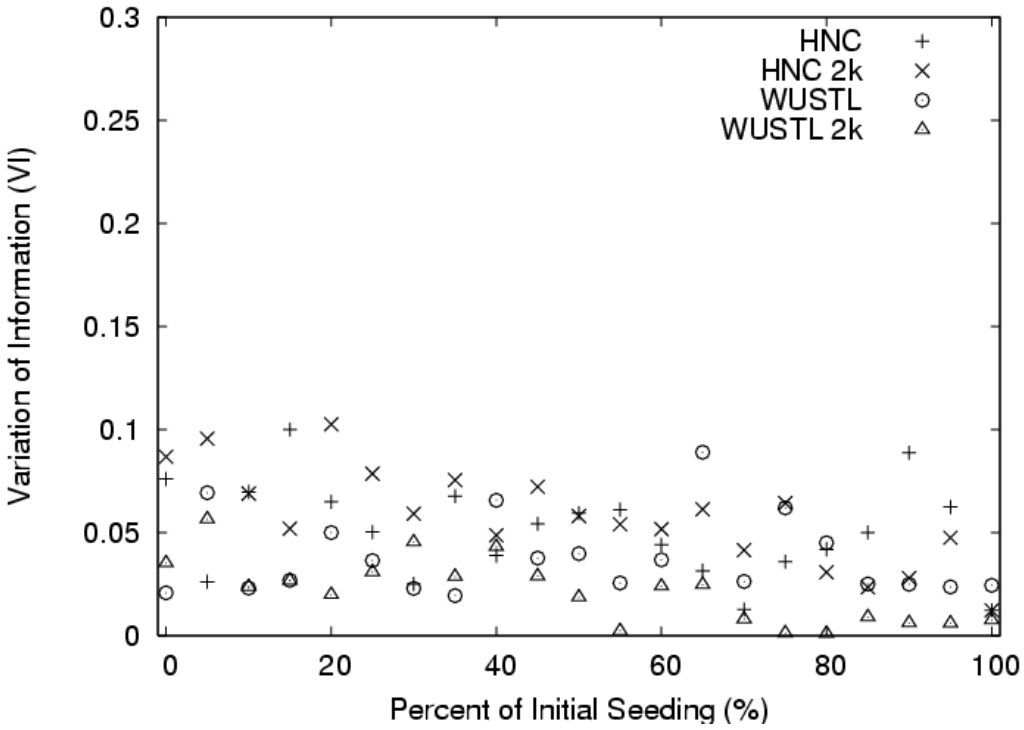


Figure 4.4: Average VI distance of concept vector reduction to full floating point K-means using the CMU20 subset one

Once again, the WMT's that used the 2k training method did not show a major improvment for the HNC WMT. The WUSTL 2k training method did show a slight improvement over the WUSTL for seeding values greater than 40 percent. The regular WUSTL WMT provided better VI distances when the seeding value was less than 40 percent.

4.6 Cosine Distance Mapping with Concept Vector Scaling

The hardware implementation of K-means uses both the cosine distance mapping and the concept vector scaling. It was expected that the combination of these methods would cause a larger deviation from the full floating point K-means software. Figure 4.5 shows that the largest average VI distance had a value around .15. This shows that the overall reduction to a hardware implementation is not significantly different from a normal K-means implementation.

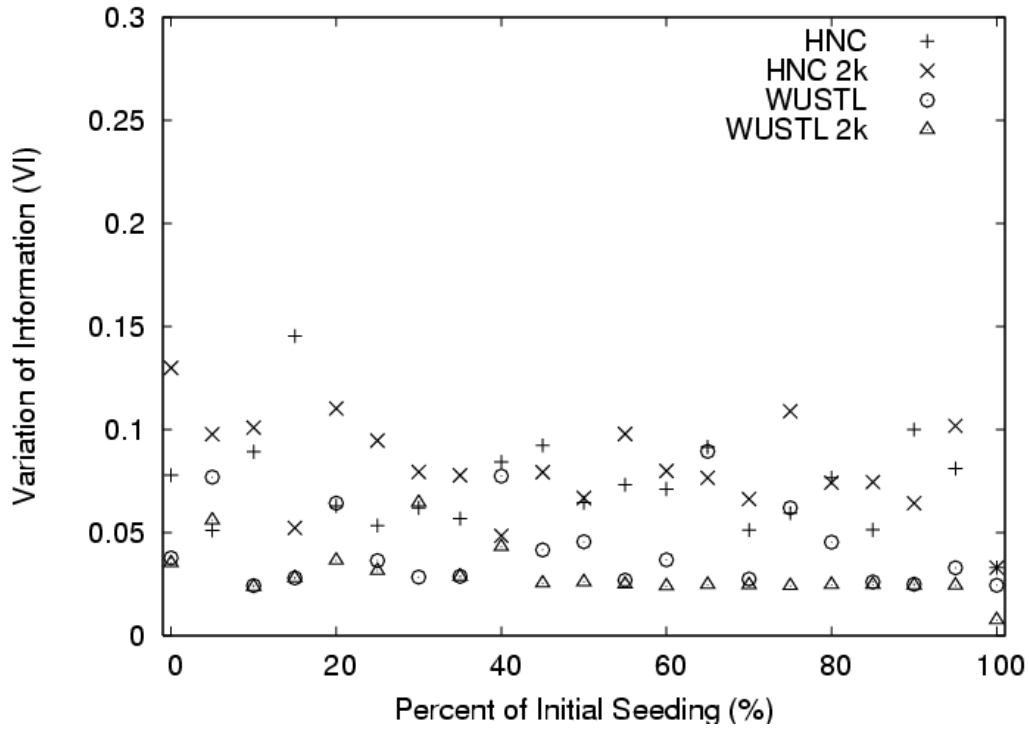


Figure 4.5: Average VI distance of the hardware simulation to full floating point K-means using the CMU20 subset one

Both WUSTL WMTs provided better VI distances than the HNC WMTs. the WUSTL WMTs were also about the same for all seedings. When they were different, the WUSTL 2k table returned better VI distances.

Chapter 5

K-means Hardware Architecture

Reconfigurable hardware allows for many levels of parallelism. The level of parallelism used for a design depends on the problem and the requirements of the solution. The amount of parallelism that can be achieved is a function of the number of documents, the number of concepts, the number of elements per document and concept, and the resolution of each element.

The most parallelism that a clustering system could achieve would occur when one computes all the distances from one document to all concept vectors at once. This level of parallelism provides high throughput, but it affects other parameters in the system. In particular the number of concept vectors that could be stored in local memory would be limited because of to the number of distance modules that can be replicated on an FPGA.

5.1 Design Objectives

The K-means clustering algorithm performs three primary operations. These operations are: calculate the distances, identifying the cluster assignments, and update the clusters. Each of these operations can be implemented in hardware. The most important aspect of porting software algorithms to hardware is to ensure that the algorithm achieves performance at a level higher than the software implementation. The most important and parallel part of the K-means algorithm is the computation of the distance metric. The other operations can be implemented in hardware with comparable speed of the software algorithm.

5.2 Design Decisions

Both the number of gates and the amount of memory available on FPGAs has increased dramatically in recent years. This increase allows for more complicated systems to be implemented in hardware. Even with the increase in memory and logic resources, there are still limitations that affect how software algorithms can be mapped to hardware.

The hardware was designed in a modular framework. This allows for more complicated operations to be created from smaller and simpler operations. For instance, the cosine distance module is created by linking together a controller, a dot product, and a normalization circuit.

The hardware was designed to cluster data vectors that consist of 4000 elements. Each of these elements have 4-bits for a representation. The centroid vectors are very similar to the data vectors, however their elements can contain an 8-bit number. The centroid values require a larger representation because they are the average of the data vectors that are contained within them.

5.3 Communications

A protocol was developed to send clustering information to the hardware. This protocol allows a networked PC to send data vectors, initial centroid vectors, and their sum of squares and assignments to the hardware. The protocol is comprised of three packet formats.

5.3.1 Packet Formats

The data packet format is shown in Figure 5.2. The data packets contain an opcode of x01 in the first byte. The next tree bytes contain the centroid assignment. The next four bytes is reserved for the data vector identifier. The last entry in the data vector packet before the data vector is the sum of squares. The sum of squares occupies four bytes. The rest of the packet contains the data vector elements.

OP	
x01	Load Data Vectors
x10	Load 8-bit Centroid Vectors
x11	Load 28-bit Centroid Vectors
x30	Start Clustering

Figure 5.1: Opcodes for clustering packets

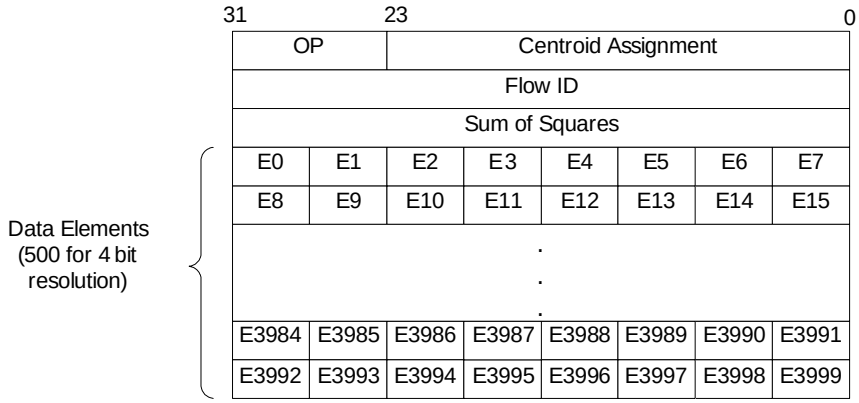


Figure 5.2: Data Packet

The 8-bit centroid vector packets contain an opcode of x10 in the first byte, followed by the number of data vectors within the cluster (three bytes). The next four bytes contain the sum of squares for the 8-bit centroid vector. The rest of the packet contains the 8-bit centroid elements.

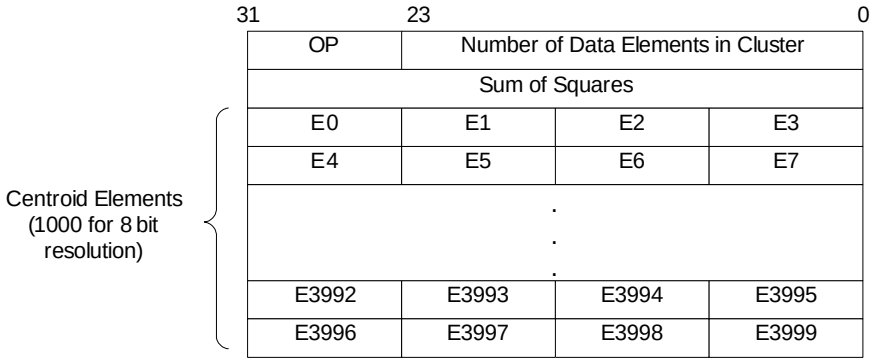


Figure 5.3: 8-bit Centroid Packet

The 28-bit centroid vector packets are very similar to the 8-bit packets. The only difference is the size of the elements being sent.

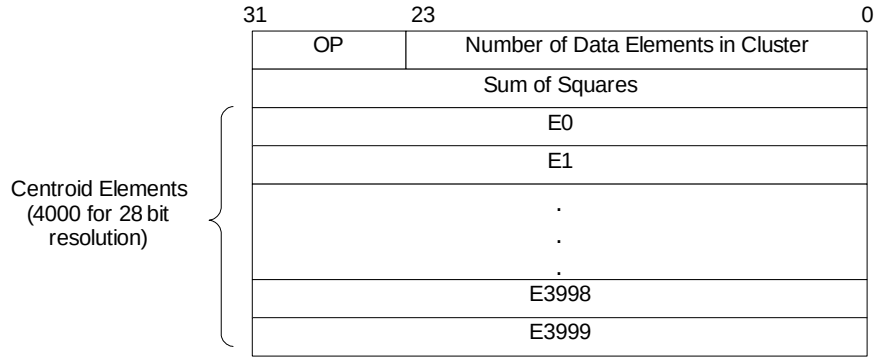


Figure 5.4: 28-bit Centroid Packet

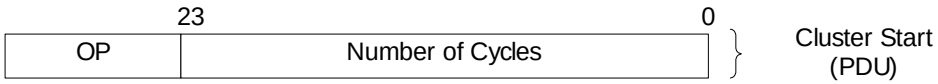


Figure 5.5: Start Packet

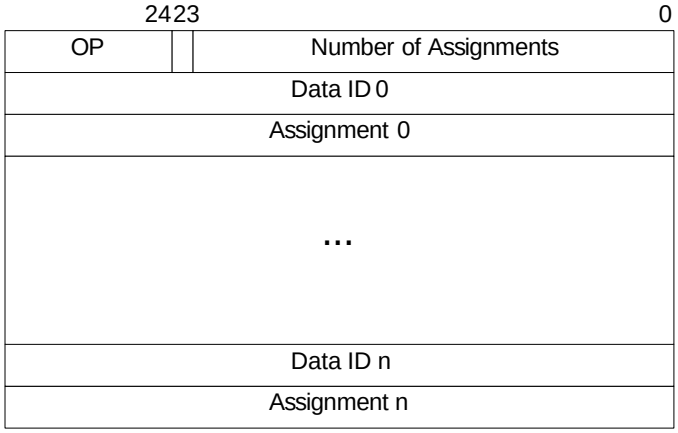


Figure 5.6: Output Packet

5.3.2 Program to Load Data in Hardware

A program was created to use the communication packets to load data into hardware and issue the command to start the clustering hardware. The program reads the data vectors and randomly assigns them to initial centroids. The sum of squares for the data vectors are then computed. The random centroid assignments are used to calculate the starting centroid vectors, the sum of squares and the number of data vectors contained in each centroid. The data vectors and their cluster assignments

are then packed into UDP packets and sent to hardware. The centroid vector data is then packed into UDP packets and sent to the hardware. After all the data is sent, the UDP packet that starts the clustering is sent.

5.3.3 Data Capture Program

Results from the K-means clustering hardware must be received and stored. This verifies that the hardware is working correctly in addition to capturing the final clustering that can be used for classification. The hardware sends the data vector identifiers and their centroid assignments after every epoch. An epoch is a full iteration through all data vectors and updating their corresponding assignment. This allows for a software system to track the convergence of the clustering hardware. The capture program receives the identifiers and the assignments and stores them into a simple XML format that can be used in post processing.

```
< ?xml version="1.0" encoding="UTF-8"?>
< ClusterOutput>
< MetaData File_Type='Cluster_Output' version='1.2' />
< assignments>
                        < flowID="0" assign="3" />
                        < flowID="1" assign="3" />
                        < flowID="2" assign="3" />
                        < flowID="3" assign="3" />
                        < flowID="4" assign="1" />
                        < flowID="5" assign="1" />
                        < flowID="6" assign="1" />
                        < flowID="7" assign="2" />
                        < flowID="8" assign="2" />
                        < flowID="9" assign="2" />
                       < flowID="10" assign="0" />
                       < flowID="11" assign="0" />
                       < flowID="12" assign="0" />
                       < flowID="13" assign="0" />
                       < flowID="14" assign="0" />
                       < flowID="15" assign="0" />
  </assignments>
 </ClusterOutput>
```

5.4 Highly Parallel Architecture

The architecture of the hardware K-means was designed with different layers. The overall hardware implementation is comprised of smaller modules that are well defined and have specific functions. The cosine distance module is comprised of a control processor, normalization module, dot product module, and a division circuit. The normalization module is comprised of two square roots and a multiply. The architecture was designed from the top down; however, the implementation of the hardware circuits were designed from the bottom up.

The architecture of the hardware K-means is comprised of three main modules: cosine distance, greedy accept, and update. These modules are comprised of smaller task oriented modules. The smaller modules allow the design to be highly adaptable in the future. The high level diagram of the K-means clustering hardware as shown in Figure 5.7.

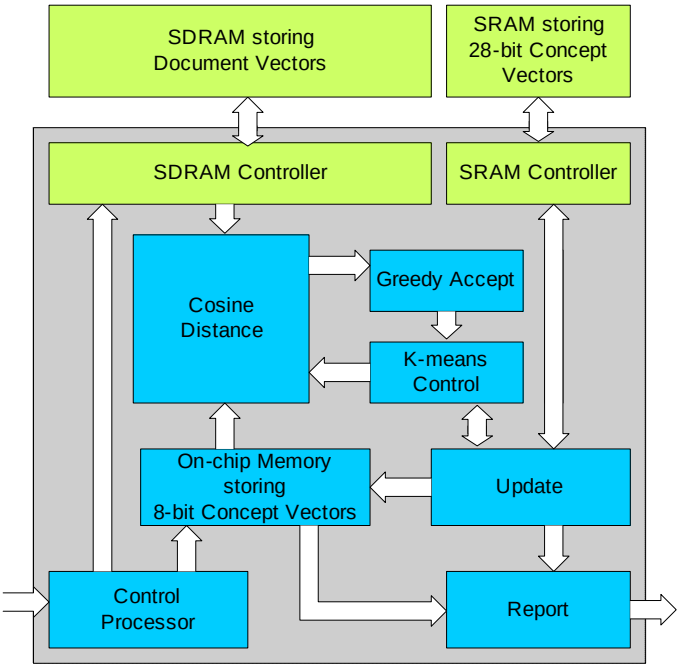


Figure 5.7: Hardware Clustering Block Diagram

5.4.1 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 is comprised of a 8-bit concept storage, normalization, dot product, division and a control module. The control module is used to control accesses to the on-chip memory that stores the 8-bit concepts. It also starts the normalization module and the dot product module. The output of the normalization and the dot product is then sent to the divide module. The divide module was created from the Xilinx Core Generator. The output of the division is a cosine distance that is mapped into the range between 0 and 255. The cosine distance circuitry can produce a cosine distance every 294 cycles (3.675 μs when running at 80 Mhz).

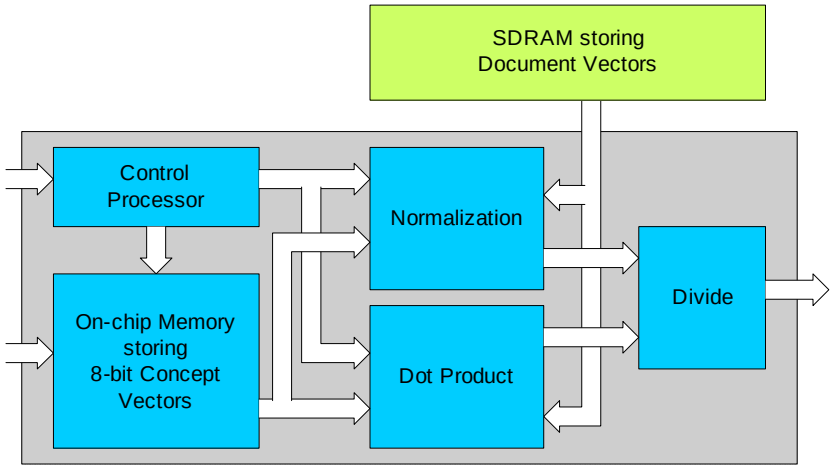


Figure 5.8: Cosine Distance Block Diagram

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.

5.4.2 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 to one in a true cosine theta calculation)

is held by two or more concept vectors, then the concept vector with the smallest index value is chosen. The number of comparisons needed to determine the best distance is equal to the number of cosine distance inputs minus one.

5.4.3 Update

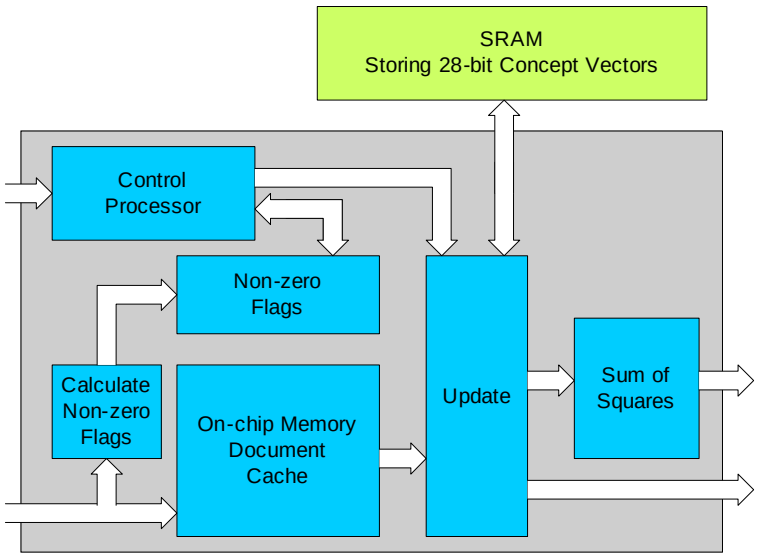


Figure 5.9: Update Block Diagram

As the document vector is streamed into the cosine distance module, the update module caches the data. This allows the update module to identify and store the number of non-zero elements. The update module then uses the non-zero element flags to load and update the concept vectors. This is useful when the document vectors are sparse.

This module loads the extended 28-bit concept vector values from memory and updates both the 28-bit and the truncated 8-bit concept vector value. The extended vector values are necessary in the update procedure to avoid the cascading precision error. If the truncated 8-bit values were used for the update, 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.

5.4.4 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. Once the control packet is sent to the load processor, it starts the clustering by signaling the K-means controller.

5.4.5 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.

5.4.6 Report

The report module sends 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.

Chapter 6

Results

6.1 Testing Environment

The FPX hardware was used to test the K-means hardware module. The K-means module was loaded into one FPX card within the GVS-1000. All other cards within the GVS-1000 were configured as pass through cards (meaning all data passed though the cards without any interruption). The GVS-1000 was connected to a computer system that sent and received all the data to the hardware circuits. The configuration can be seen in Figure 6.1. The connected computer sends all the UDP packets to load data vectors, 8-bit centroid vectors, and 28-bit centroid vectors. Once the data was loaded, a command was sent to start the clustering. After the hardware computed the results, packets with the results were sent to the attached PC.

6.2 Experiments

The hardware experiments used the CMU 20 newsgroup data set (CMU20) [23]. Each document in the data set was converted into a 4000 dimensional data vector by using a word mapping table (WMT). A discussion of WMTs can be seen in Section 2.6. The WMTs used in the experiments had two important characteristics: the first was that the features which overflowed the 4-bit counter were saturated to a value of fifteen; the second characteristic was how the WMTs mapped the words. The first two thousand elements of the WMT were used to map words found in the training set. The last two thousand elements were reserved for the unseen words. The theory is that the last two thousand elements would provide a better clustering of unknown concepts.

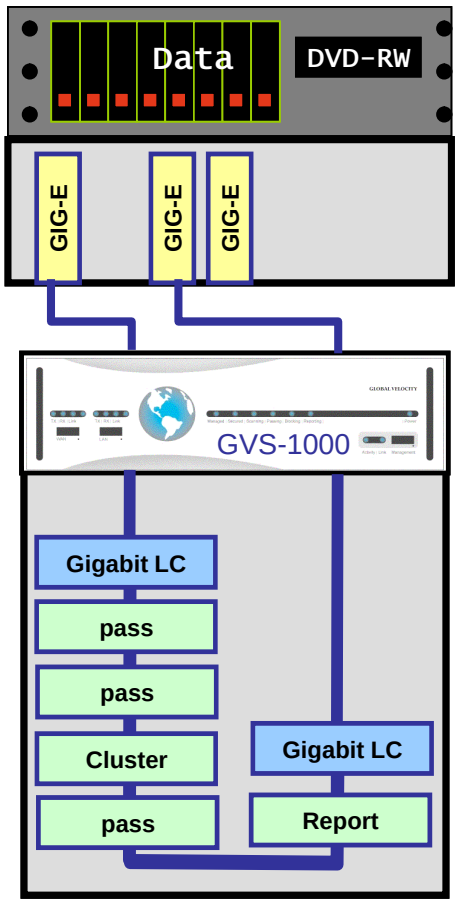


Figure 6.1: Hardware configuration used for running experiments

6.3 Clustering Running Times

When comparing clustering algorithms the speed of computation is an important metric. Since the distance computation dominated most of the CPU running time, data was gathered to show the time required to produce the distances for one document. The hardware implementation was compared to three software versions of K-means. The software varied the precision of the concept vectors. The character implementation used an 8-bit representation. The integer used a 16-bit representation, and the double implementation was a full floating point representation.

Figure 6.2 shows the average running time for comparing one document to all K cluster centroids. The character and integer representations are very close in the running time. The double representation performs slower than all other implementations, which makes sense given that the double uses a higher level of precision. The software implementations run sequentially. So, as the number of concepts is changed from 4

to 10 to 25, the software implementation speed decreases. The parallel hardware implementation runs at the same speed regardless of the increase in the number of concepts. On the current FPX hardware, the maximum amount of parallel concepts is limited to four. So the important comparison is between the values of $K = 4$. Even though the hardware has significant gains over the double representations, the best comparison is gainst the character representation.

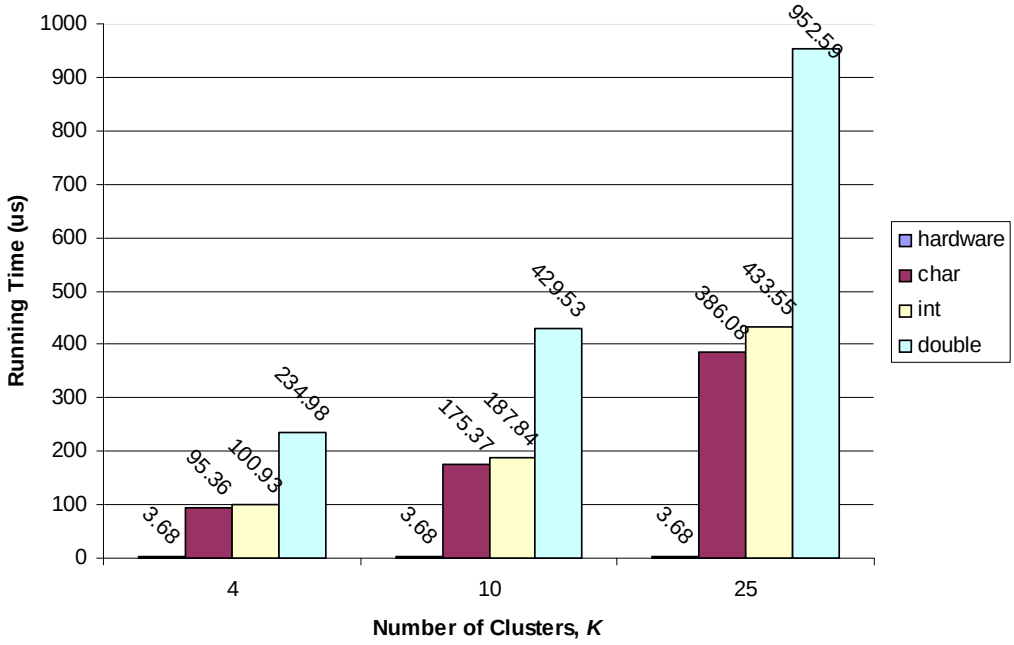


Figure 6.2: Average time for each software implementation and the hardware implementation comparing one document to all k concepts

6.4 Clustering Speed for Cosine Distance

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 software implementation that runs in 95.36 μs with four clusters and clustering hardware running at 80 MHz, the hardware is 25.95 times faster (Figure 6.3). As more clusters are added and the hardware clock speed is increased, the speed gain of the hardware increases. When the clustering

system is implemented on the Virtex4 LX200 FPGA using a clock frequency of 250 MHz, the hardware is 329 times faster then software.

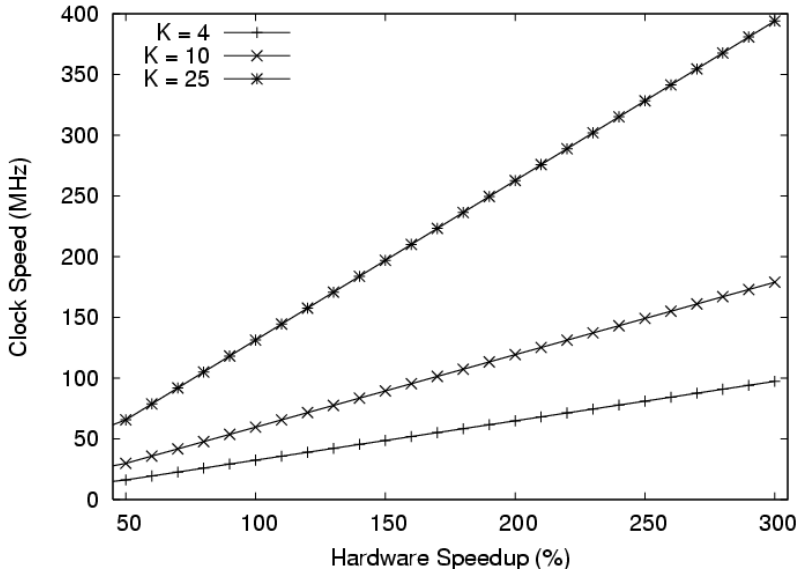


Figure 6.3: Hardware Speedup in comparison to the character implementation in software as clock frequency increases

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 is 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 6.1 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.

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 6.1: Device utilization for Hardware K-means with Cosine Theta Distance metric using four concepts across different platforms

Chapter 7

Conclusions

A high speed, parallel K-means algorithm has been synthesized for both a Xilinx VirtexE 2000 FPGA available on the latest generation of the open prototype platform and a Xilinx Virtex4 LX200. 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 much higher rates of clustering than using software algorithms.

The effect of using an integer approximation of the cosine theta distance metric, and reducing the size of the representation for the concept vectors were explored. When compared to a full K-means implementation using the same initial assignments, it was shown that the VI distance between the resulting clusterings was very small. The cosine distance mapping did not significantly impact the clusterings. The reduction of the centroid representation did cause variations in the clustering performance. When combined, the cosine mapping and the concept vector approximation returned VI distances that were only slightly worse than the concept vector reduction alone.

By scaling values to make best use of the 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 it is in software. This design of clustering in hardware maintains its performance advantage even as the number of concepts increases. We observed 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) by a factor of twenty-six.

7.1 Future Work

7.1.1 Hardware Adaptation

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 that the algorithm finds the global maximum. Simulated annealing enables the algorithm to jump out of local maximum by probabilistically accepting worse 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 solution.

A semi-parallel architecture would make best use of hardware and software resources. The size of the circuit in hardware could scale with the number of distance metrics that fit within the FPGA. This hybrid architecture is a variation of the full parallel architecture. The difference resides in the addition of a concept vector controller. This controller would load in the first N concepts into block memory. When the distances for the first N concepts are calculated, the next N concepts are loaded and used for the distance calculation. After all of the distances are computed, a decision is made to determine if a document should have a new assignment. The process of updating the concept vectors is the same as the full parallel architecture.

7.1.2 Porting to other FPGAs and Platforms

We have already begun porting the implementation to newer FPGAs. This allows the circuit to achieve higher speed gains without a major modification to the architectural design. Implementing the full parallel architecture on a Virtex4 allows for an increase in the number of parallel cosine distance metrics. In addition, the Virtex4 contains DSP units that would allow for very efficient implementation of the distance circuits. A direct port of the circuit to the Virtex4 would allow for an increase in the number of concepts; however, design enhancements of the distance metrics would enable massive performance gains by using the DSP blocks.

Other platforms, such as the Stanford NetFPGA, allow for a very low-cost implementation of circuits using a fully open hardware platform. The NetFPGA is a PCI-based card that contains two external SRAM banks, a Virtex2 Pro 30 FPGA, four Gigabit

ethernet devices, and two high speed serial interfaces. The total cost of the board is less than $400. High speed interfaces would allow communication to occur between multiple NetFPGA boards. This communication could give the clustering implementation higher performance when needed.

The PCI interface in the NetFPGA board allows for easy implementation of the semi-parallel architecture. Software can run on the Linux host computer while hardware runs on the PCI device. The hybrid platform would allow us to offload portions of the clustering code to the FPGA. Since the Virtex2 Pro has an embedded PowerPC Processor, software could also be run in the PowerPC embedded in the FPGA. Video adapters are readily available for the PCI bus that are equipped with Graphic Processing Units (GPUs). Prototype boards with cell processors have also been built to attach to a system PCI bus. A complete system could be readily integrated containing a mix of software, FPGA hardware, GPUs, and Cell Processors.

Appendix A

Hardware Schematics

A.1 K-means

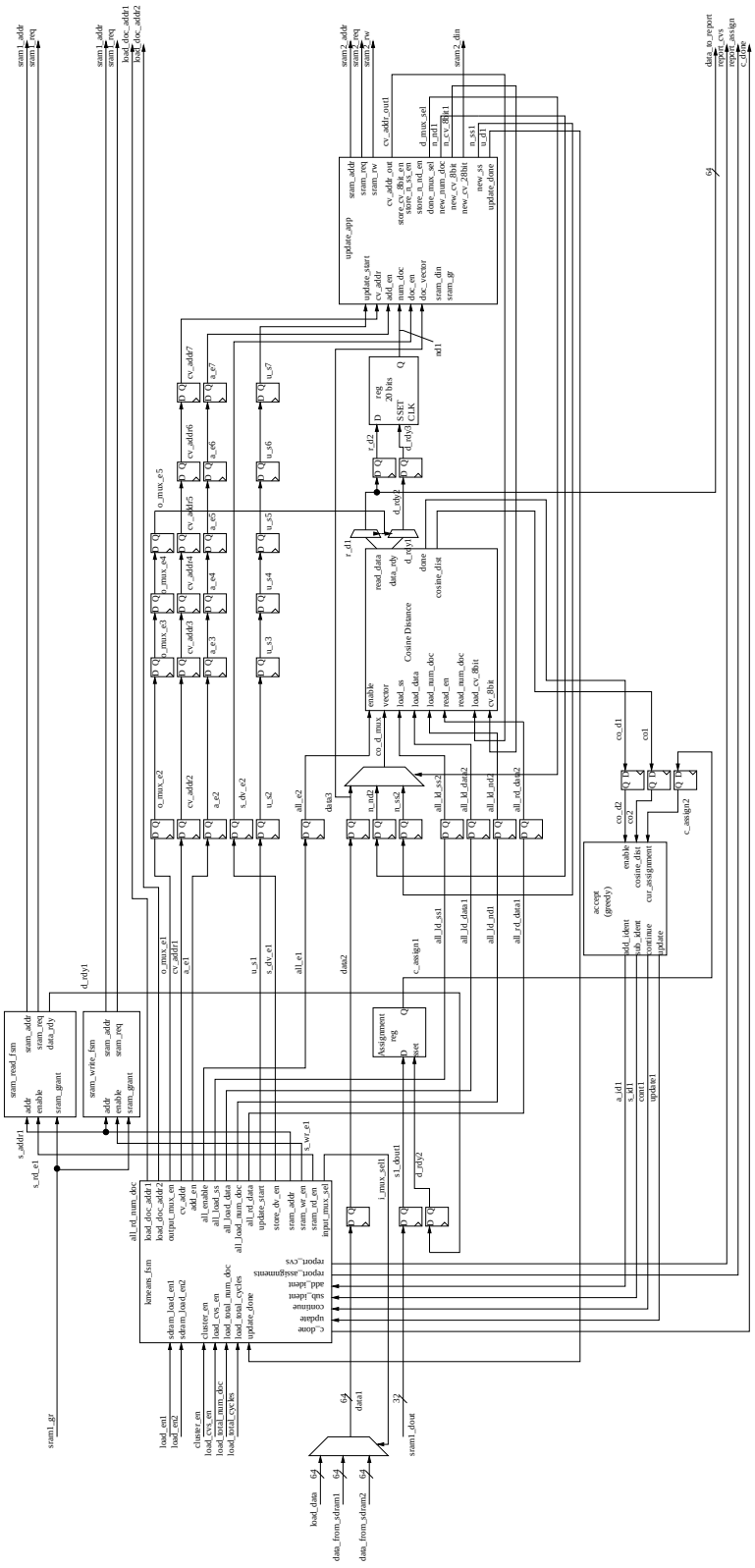


Figure A.1

A.2 Cosine Distance

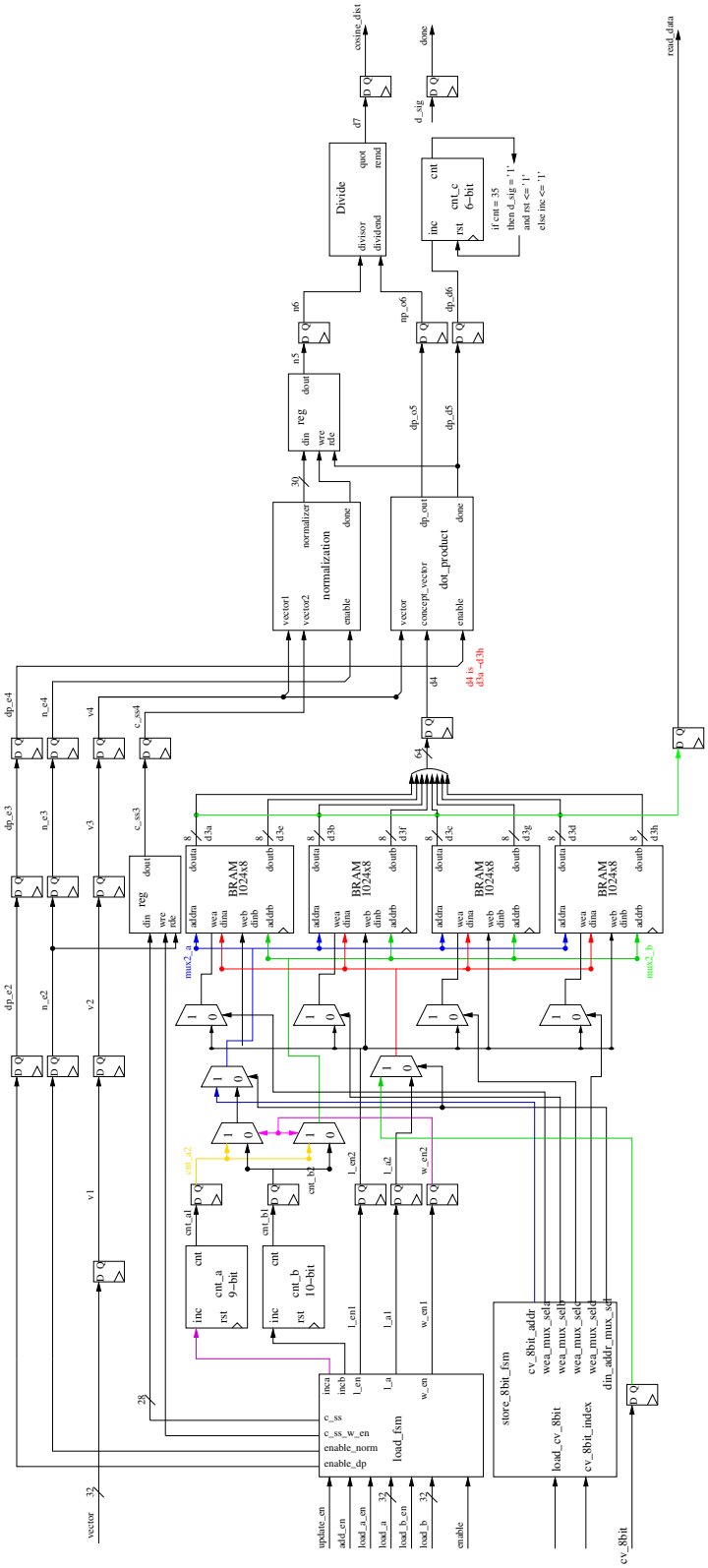


Figure A.2

A.3 Normalization

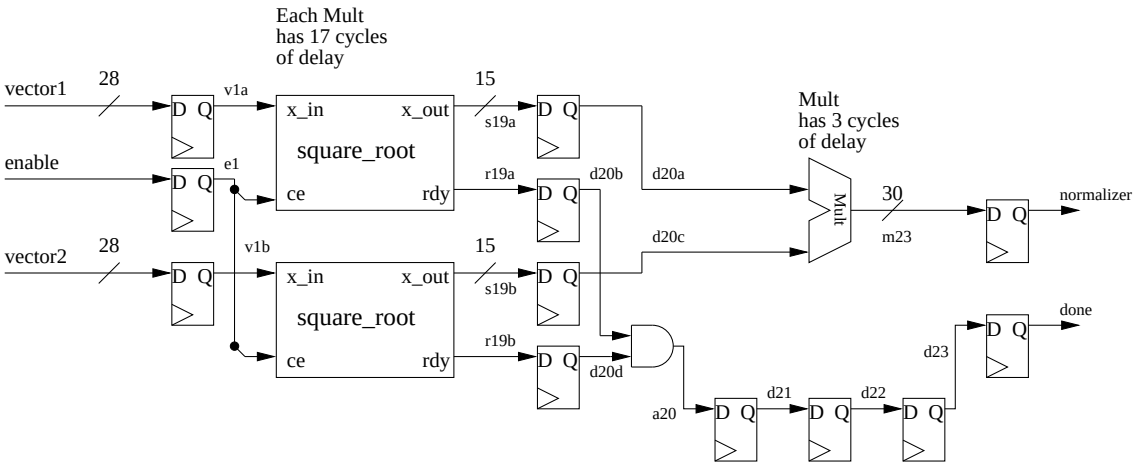


Figure A.3

A.4 Greedy Accept

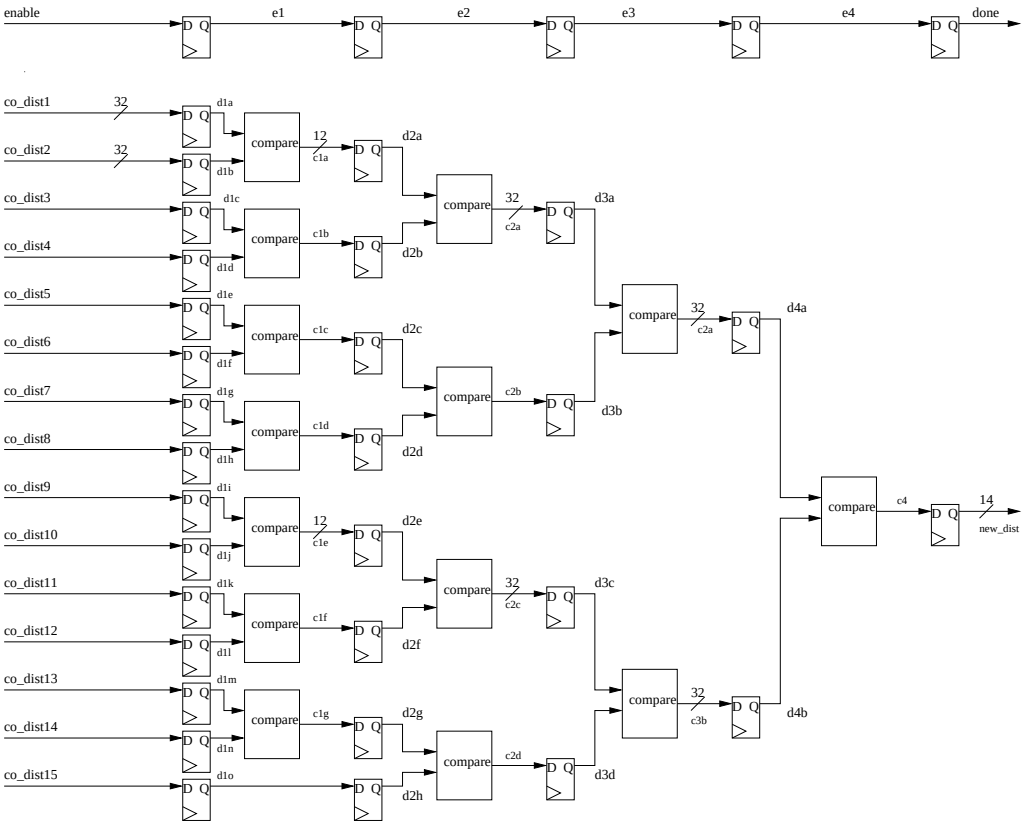


Figure A.4

A.5 Manhattan Distance

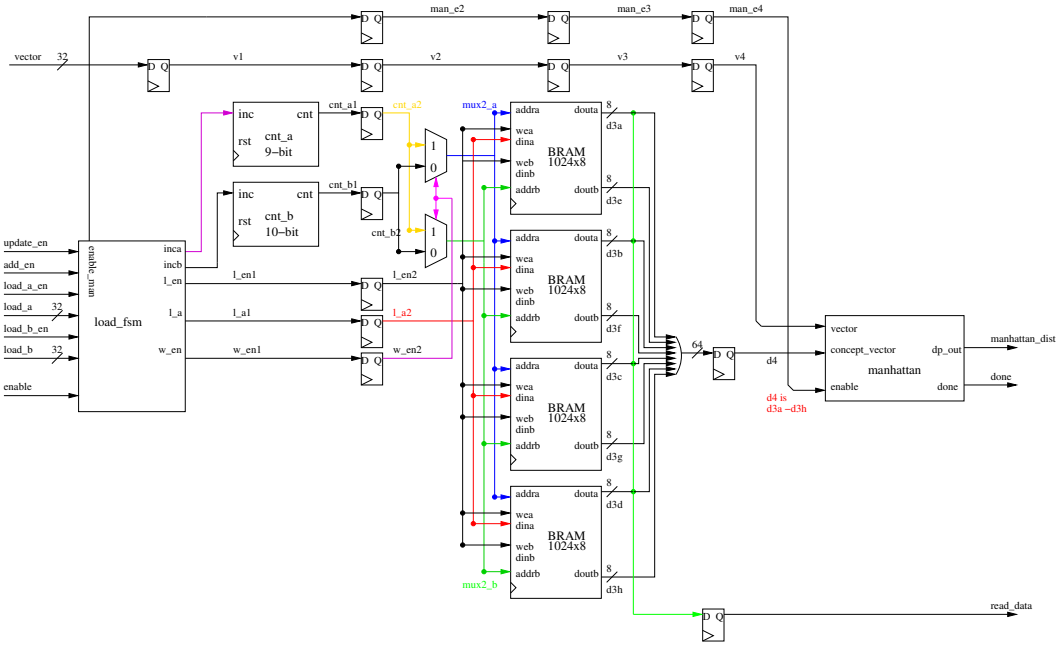


Figure A.5

A.6 Manhattan Calculation

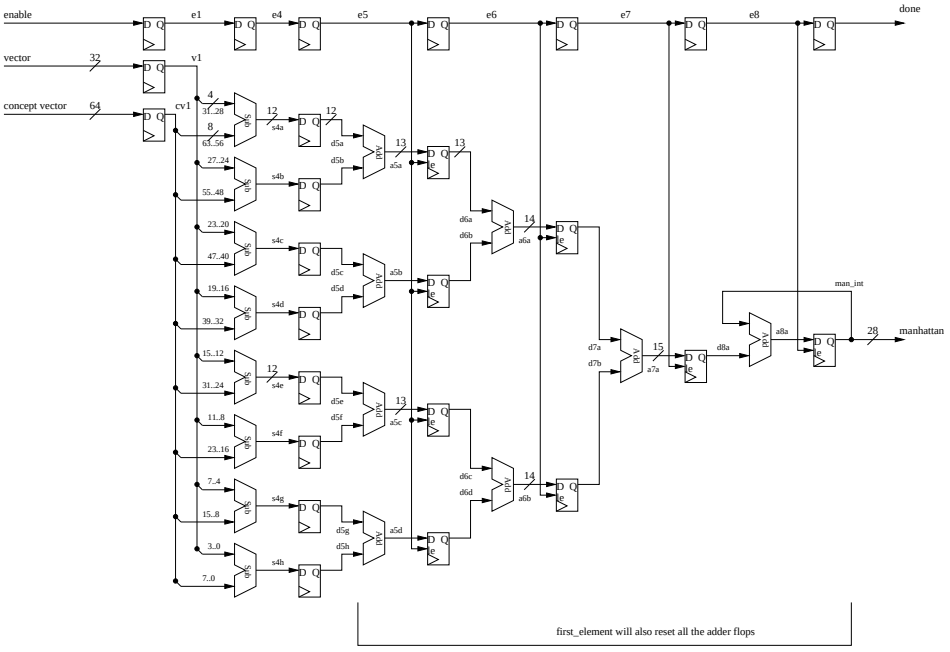


Figure A.6

A.7 Update

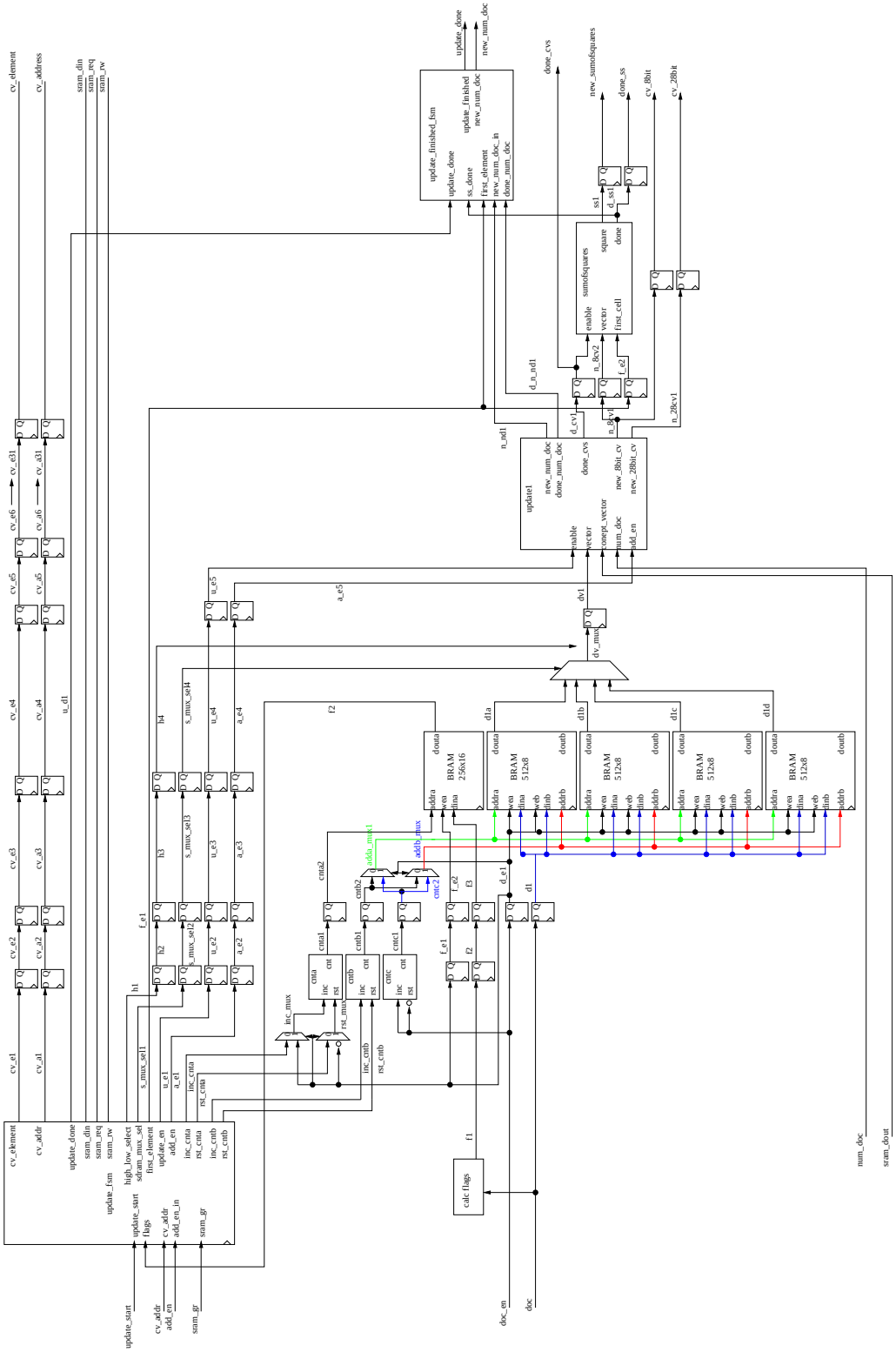


Figure A.7

Appendix B

Semi-Parallel Hardware Architecture

In order to implement a balanced system that provides a flexible structure and allows for a wide range of clustering parameters, semi-parallel architectures were considered.

B.1 Semi-Parallel Architecture

Although the fully parallel architecture achieves the highest throughput, the architecture has drawbacks. The main problem is that the number of concepts that can be implemented on an FPGA is limited by the number of distance metrics that can be replicated in hardware. This does not scale well. An alternative, semi-parallel architecture allows the hardware to scale with the number of distance metrics that fit on an FPGA independently of the number of concept vectors that can be calculated.

The semi-parallel architecture allows for a set of concept vectors to be loaded into the on-chip memory. The distance from a document to the set of concept vectors is then calculated. After the last element of the document is sent to the distance metric module, a new set of concept vectors is loaded into the on-chip memory. This allows the number of concepts to be limited to only the number that can be fit into memory (SRAM). The number of concept vectors within a set is determined by the number of distance modules that can be replicated on the FPGA. For the Xilinx VirtexE 2000 the number of distance metrics that can be replicated is four. The Xilinx Virtex4 LX 200 can supports 25 distance metric modules. This means that on the Virtex4 the system can calculate 25 distances at once, and support larger numbers of concept vectors. The Semi-Parallel architecture is shown in Figure B.1.

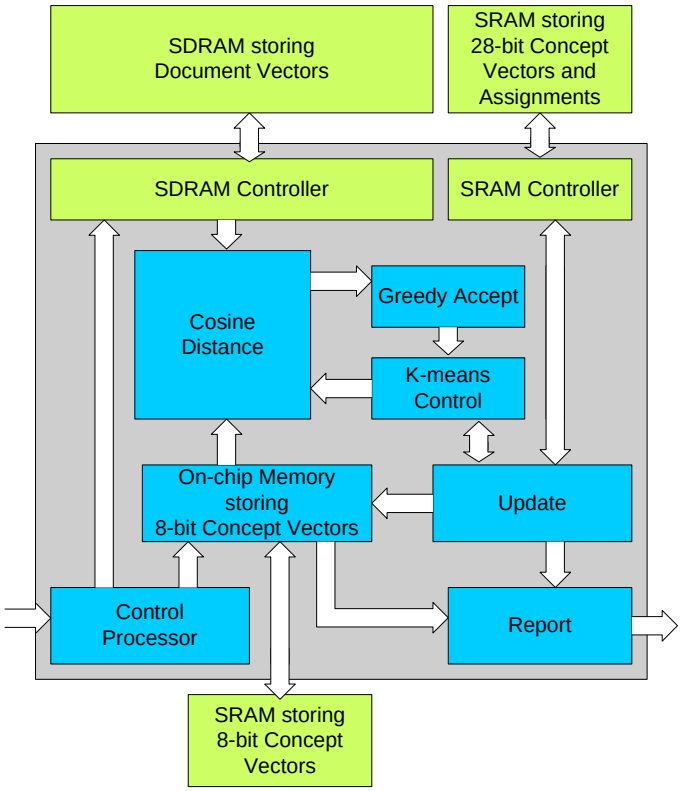


Figure B.1: Semi-Parallel Hardware Clustering Block Diagram

The design is similar to the Fully Parallel architecture. The difference resides in the K-means module. It is necessary to implement a control module that reads the eight-bit concept vectors from SRAM and populate the on-chip memory. In a greedy implementation without simulated annealing, there is no need to cache all the distances that are calculated. The greedy accept module only needs to be as parallel as the number of distance metrics. After each set distances are calculated, only the best distance needs to be cached and compared to the next set of distances.

References

[1] M. Steinbach, G. Karypis, and V. Kumar, "A Comparison of Document Clustering Techniques," in *Proceedings of the 6th KDD Workshop on Text Mining*, Boston, Massachusetts, Aug. 2000.

[2] R. Duda and P. Hart, *Pattern Classification and Scene Analysis.* John Wiley and Sons, Mar. 1973.

[3] 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

[4] 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

[5] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," 2001. [Online]. Available: citeseer.ist.psu.edu/ng01spectral.html

[6] R. Rohwer and D. Freitag, "Towards full automation of lexicon construction," in *Human Language Technology/North American chapter of the Association for Computational Linguistics*, Boston, Massachusetts, 2004.

[7] I. S. Dhillon, S. Mallela, and D. S. Modha, "Information-theoretic co-clustering," in *Proceedings of The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, Aug. 2003.

[8] D. L. Wallace, "Comment," in *Journal of the American Statistical Association*, 1983.

[9] E. B. Folwkes and C. L. Mallows, "A method for comparing two hierarchical clusterings," in *Journal of the American Statistical Association*, 1983.

[10] M. Meilă, "Comparing clusterings," 2003. [Online]. Available: citeseer.ist.psu.edu/meila02comparing.html

[11] L. Hubert and P. Arabie, "Comparing partitions," in *Journal of Classification*, 1985.

[12] B. Larsen and C. Aone, "Fast and effective text mining using linear time document clustering," in *Proceedings of the conference on Knowledge Discovery and Data Mining*, San Diego, California, 1999.

[13] M. Meilă and D. Heckerman, "An experimental comparison of model-based clustering methods," in *Machine Learning*, 2001.

[14] 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.

[15] C. M. Kastner, G. A. Covington, A. A. Levine, and J. W. Lockwood, "HAIL: A hardware-accelerated algorithm for language identification," in *15th Annual Conference on Field Programmable Logic and Applications (FPL)*, Tampere, Finland, Aug. 2005.

[16] 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.

[17] 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.

[18] D. V. Schuehler, J. Moscola, and J. W. Lockwood, "Architecture for a hardware-based, TCP/IP content-processing system," *IEEE Micro*, vol. 24, no. 1, pp. 62–69, Jan. 2004.

[19] 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.

[20] S. G. Eick, J. W. Lockwood, R. Loui, A. Levine, J. Mauger, D. J. Weishar, A. Ratner, and J. Byrnes, "Hardware accelerated algorithms for semantic processing of document streams," in *IEEE Aerospace Conference*, Big Sky, Montana, Mar. 2007.

[21] G. A. Covington, C. L. Comstock, A. A. Levine, J. W. Lockwood, and Y. H. Cho, "High speed document clustering in reconfigurable hardware," in *16th Annual Conference on Field Programmable Logic and Applications (FPL)*, Madrid, Spain, Aug. 2006, pp. 411–417.

[22] 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

[23] (2005) Cmu 20 newsgroups. [Online]. Available: http://people.csail.mit.edu/jrennie/20Newsgroups/

[24] F. Braun, J. W. Lockwood, and M. Waldvogel, "Layered protocol wrappers for internet packet processing in reconfigurable hardware," Washington University

in Saint Louis, Department of Computer Science, Tech. Rep. WU-CS-01-10, June 2001.

[25] J. Theiler, M. Leeser, M. Estlick, and J. Szymanski, "Design issues for hardware implementation of an algorithm for segmenting hyperspectral imagery," 2000. [Online]. Available: citeseer.ist.psu.edu/theiler00design.html

Vita

Gerald Adam Covington

Date of Birth May 20, 1981

Place of Birth Lexington, South Carolina

Degrees B.S. Computer Engineering, Western Michigan University, April 2003
M.S. Computer Engineering, Washington University in St. Louis, Expected December 2006

Publications

Streaming Hierarchical Clustering for Concept Mining, by Moshe Looks, Andrew Levine, G. Adam Covington, Ronald P. Loui, John W. Lockwood, Young H. Cho, *IEEE Aerospace Conference*, Big Sky, MT; March 3-10, 2007.

High Speed Document Clustering in Reconfigurable Hardware, by G. Adam Covington, Charles L.G. Comstock, Andrew A. Levine, John W. Lockwood, Young H. Cho, *16th Annual Conference on Field Programmable Logic and Applications (FPL)*, Madrid, Spain; August 28-30, 2006.

HAIL: A Hardware-Accelerated Algorithm for Language Identification, by Charles M. Kastner, G. Adam Covington, Andrew A. Levine, John W. Lockwood, *15th Annual Conference on Field Programmable Logic and Applications (FPL)*, Tampere, Finland; August 24-26, 2005.

A 3-axis acceleration sensor data acquisition instrument system, by Asumadu, J.A.; La Belle, V.; Od'Neal, R.; Covington, G. A; *Instrumentation and Measurement Technology Conference, 2004. IMTC 04. Proceedings of the 21st IEEE*, Como, Italy; May 18-20, 2004.

December 2006