Algorithmic Transformations in the Implementation of K-means Clustering on Reconfigurable Hardware

Mike Estlick, Miriam Leeser
Department of Electrical and Computer Engineering
Northeastern University, Boston, MA
{mestlick, mel}@ece.neu.edu

James Theiler, John J. Szymanski
Space and Remote Sensing Sciences Group
Los Alamos National Lab,
Los Alamos, NM
{jt, szymanski}@lanl.gov

ABSTRACT

In mapping the k-means algorithm to FPGA hardware, we examined algorithm level transforms that dramatically increased the achievable parallelism. We apply the k-means algorithm to multi-spectral and hyper-spectral images, which have tens to hundreds of channels per pixel of data. K-means is an iterative algorithm that assigns assigns to each pixel a label indicating which of K clusters the pixel belongs to.

K-means is a common solution to the segmentation of multi-dimensional data. The standard software implementation of k-means uses floating-point arithmetic and Euclidean distances. Floating point arithmetic and the multiplication-heavy Euclidean distance calculation are fine on a general purpose processor, but they have large area and speed penalties when implemented on an FPGA. In order to get the best performance of k-means on an FPGA, the algorithm needs to be transformed to eliminate these operations. We examined the effects of using two other distance measures, Manhattan and Max, that do not require multipliers. We also examined the effects of using fixed precision and truncated bit widths in the algorithm.

It is important to explore algorithmic level transforms and tradeoffs when mapping an algorithm to reconfigurable hardware. A direct translation of the standard software implementation of k-means would result in a very inefficient use of FPGA hardware resources. Analysis of the algorithm and data is necessary for a more efficient implementation. Our resulting implementation exhibits approximately a 200 times speed up over a software implementation.

1. INTRODUCTION

Multispectral and hyperspectral image data are increasingly available from a variety of sources, including commercial and government satellites, as well as airborne and ground-based sensors. This is accompanied by an increase in spatial resolution as well as an increase in the number of spectral channels. Multispectral images can have from a few to a few tens of channels per pixel, while hyperspectral data contains hundreds of spectral channels per pixel. In this paper we apply the k-means unsupervised clustering algorithm to AVIRIS data sets [8] and simulated MTI (Multispectral Thermal Imager) data sets. A single AVIRIS image contains 614×512 pixels with 224 16-bit channels, or approximately 140 MB of data. The MTI data sets are also 614×512 pixels, but have 10 channels of 16 bit data.

The image analyst's challenge is to identify the important and useful features in the image without being overwhelmed by the sheer volume of the data. One response to this challenge is provided by algorithms which segment the image by clustering pixels into classes based on the spectral similarity of each pixel to other members of the class. As well as providing the analyst with a picture summarizing the spatial organization of the different spectral types, these clustering algorithms also provide a very real compression of data. Each pixel in the image is represented by a pointer to the spectral class associated with that pixel. For 8 classes, 3 bits specifies the classification of each pixel, and since each class is identified with a cluster center which approximates the pixel's spectrum, this serves as an approximation to the whole image. This approximate image can be represented by 75K bytes – a compression factor of over three orders of magnitude for an AVIRIS data set.

As well as reducing the data for quicklook views, clustering also provides an organization of the data that can be useful for further downstream processing [7]. Several authors have shown that clustering the data beforehand increases the performance of algorithms which attempt to "learn" features from a small number of examples [2, 6]. Schowengerdt [9] suggests the use of image segmentation for change detection: a change in the segmentation is more likely to indicate an actual change on the ground, since the segmentation is relatively robust to changes in sensor performance and atmospheric conditions. It has also been demonstrated that the matched-filter detection of weak spectral signatures in cluttered backgrounds can be enhanced by first clustering the background and then employing a separate matched filter for each spectral class [4]; since the within-class variance is generally much smaller than the variance over the whole image, the within-class signal-to-clutter ratios can be improved by treating individual clusters separately.

Although there are clear advantages to clustering high-dimensional data sets, workstation implementations of clustering algorithms are notoriously slow. Most algorithms, such as k-means, are iterative and require many passes through the data before convergence is achieved. Each iteration requires a computation of distance from every data point to every cluster center, and each distance requires a calculation involving every spectral channel. This type of computation lends itself to implementation in reconfigurable hardware, where the inherent parallelism of the algorithm can easily be exploited. An FPGA implementation also allows us the flexibility to consider variants of the algorithm as well as of its implementation. FPGAs are particularly well suited to this application because the amount of parallelism and processing element bitwidths can adapt to the task, allowing the designer to take maximum advantage of the hardware at hand.

In general, implementing an algorithm in hardware involves a different set of design tradeoffs than implementing the same algorithm in software. For example, a software implementation may attempt to employ intelligent branching and extrapolations to avoid some computations or to reduce the number of iterations. But in hardware, it is often more productive to simplify the underlying operations as much as possible both to speed up the calculations and to be able to provide more parallelism. A simpler operation translates to a smaller area datapath which means that more versions of the datapath can be replicated on the chip. Ideally, a good design will employ both kinds of optimizations with only the necessary computations performed, and those performed in a massively parallel circuit.

In this paper we discuss algorithm level transforms that enable k-means to be implemented in hardware. These include alternative distance metrics and truncation. We argue that these types of transforms are necessary for making the best use of FPGA hardware. We also present a detailed look at our hardware implementation with timing results.

In the next section we present the k-means algorithm. Next we present a discussion of algorithmic transforms. In Section 4 we present our current implementation of the k-means algorithm on the Annapolis Wildstar board.

2. K-MEANS CLUSTERING

Given a set of N pixels, each composed of D spectral channels, and represented as a point in D-dimensional Euclidean space (that is, $\mathbf{x}_n \in \mathcal{R}^D$, with $n = 1, \dots, N$); we partition the pixels into K clusters with the property that pixels in the same cluster are spectrally similar. Each cluster is associated with a "prototype" or "center" value which is representative of (and close to) the pixels in that class. One measure of the quality of a partition is the within-class variance; this is the sum of squared (Euclidean) distances from each pixel to that pixel's cluster center.

For a fixed partition, the optimal (in the sense of minimum within-class variance) location for each center is the mean of all pixels in each class. For a fixed choice of centers, the optimal partition assigns points to the cluster whose centers are closest. The k-means clustering algorithms (there are several variants) provide an iterative scheme that operates over a fixed number (K) of clusters, while attempting to simultaneously optimize center locations and pixels assignments.

From an initial sampling, the algorithm loops over all the data points, and reassigns each to the cluster whose center it is closest to. After the pass through the data, the cluster centers are recomputed. Note that other variants of k-means update the cluster centers each time a point is reassigned to a new cluster. This leads to faster convergence, but is more difficult to implement in hardware. Each iteration reduces the total within-class variance for the clustering, so it is guaranteed that after enough iterations, the algorithm will converge, and further passes will not reassign points. It bears remarking that this is only a local minimum. There may be an assignment of pixels to classes which produces a smaller within-class variance, but to search all possible assignments (there are of order $K^N/K!$ of them) would be an impossibly large task for all but the smallest values of N.

3. ALGORITHMIC TRANSFORMS

We investigated two major algorithmic transforms in our FPGA implementation of k-means. The first was the use of alternative distance measures and the second was the truncation of the input data and cluster centers.

3.1 Alternate Distance Measures

Points are assigned to the cluster centers to which they are closest; for the minimum-variance criterion, "closest" is defined in terms of the Euclidean distance. Consider a point $\mathbf{x}$ and cluster center $\mathbf{c}$ where i indexes the spectral components of each. The Euclidean distance is defined:

$$\|\mathbf{x} - \mathbf{c}\|^2 = \sum_i |x_i - c_i|^2 \tag{1}$$

Other distance measures can also be used; for instance, the general family of p-metrics (for which the Euclidean distance is the special case $p = 2$) is given by:

$$\|\mathbf{x} - \mathbf{c}\|^p = \sum_i |x_i - c_i|^p \tag{2}$$

To perform a k-means iteration, one must compute the distance from every point to every center. If there are N points, K centers, and D spectral channels, then there will be $O(NKD)$ operations. For the Euclidean distance, each operation requires computing the square of a number.

The Euclidean distance has several advantages. For one, the distance is rotationally invariant. Furthermore, minimizing the Euclidean distance minimizes the within-class variance. On the other hand, the Euclidean distance is more expensive than the alternatives that we are considering. The Manhattan distance, corresponding to $p = 1$, is the sum of absolute values of the coordinate differences; the Max distance, corresponding to $p = \infty$ is the maximum of the absolute values of the coordinate differences. In hardware, calculating the Euclidean distance would be significantly slower than calculating the Manhattan distance. This is due to the fact that a multiplication is required for every channel and every cluster per pixel, so the amount of parallelism that can be exploited in the hardware implementation would decrease drastically. The Manhattan distance is approximately twice as fast in software than Euclidean, but significantly faster in hardware.

Neither of the alternative distances requires any multiplication, and the Max distance has the slight advantage that the number of bits required to express the total distance does not increase with the dimension D. For the Manhattan distance, we require $\log_2 D$ extra bits for the total distance over and above the bits required for the coordinate differences.

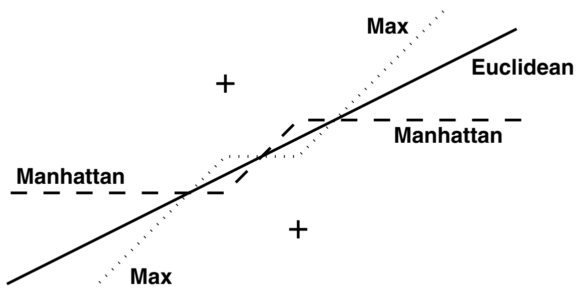

Figure 1: Decision boundaries using different distance metrics.

We examine effects of truncation on intermediate values as well as on input data in the next section.

We performed data independent and data dependent experiments to determine if the Manhattan or Max distance measures were acceptable for k-means clustering [10].

The data independent experiments estimate how often points would be mis-assigned because a cheaper distance metric was used. A correct assignment is defined to be that chosen by the Euclidean metric. The effect of metric choice on the quality of a clustering depends on many factors: the number of clusters, the dimension of the space, and the nature of the data in that space. To eliminate as many of these factors as possible, we considered an idealized situation in which three points are placed at random on the surface of a D-dimensional sphere. Two of the points are taken to be cluster centers, and the third is a data point. We compute the Euclidean distances from the data point to the two centers in order to determine which is truly closer. Then we compute the distances from the data point to the two centers using an alternative metric (Manhattan, Max, or linear combination thereof), and note whether this second pair of distances correctly identified the closest center. From a set of 10^5 such trials, we compute two statistics: relative variance and misclassification rate. The relative variance is the ratio of within-class variance for the cluster assignments provided by the alternative metric, divided by the within-class variance that would have been obtained if Euclidean distances had been used to assign points to centers. That the value is always larger than one reflects the fact that the Euclidean distance is the optimal choice for minimizing within-class variance. We estimated the misclassification rate by counting the fraction of trials for which the cheaper metric assigned the point incorrectly.

In addition to Manhattan and Max distances, we also considered a linear combination of the two. Consider Figure 1. The decision boundaries for Euclidean, Max and Manhattan are shown, where the points marked by + signs are cluster centers. A linear combination of Max and Manhattan metrics should more closely approximate the Euclidean metric than either taken alone. This linear combination is defined:

$$\|\mathbf{x} - \mathbf{c}\| = \alpha \max |x_i - c_i| + (1 - \alpha) \sum_i |x_i - c_i| \tag{3}$$

Here α ranges from 0 to 1. If α is chosen to be a negative power of 2, this linear combination can be implemented with little hardware overhead.

In Fig. 2, we plot the relative variance and misclassification rate as a function of the number D of spectral channels. A misclassification occurs when the center that is closest according to the Euclidean metric is not closest according to one of the other metrics. Note that although the relative variance is decreasing for large values of D, the rate of misclassification is monotonically increasing. For the Manhattan metric (and for the $\alpha = 0.25$ linear-combination metric), the misclassification rate saturates at about fifteen percent, but for the Max metric, the error rate begins to approach fifty percent. That is, for very large dimension D, the Max metric is not much better than just assigning points to clusters at random.

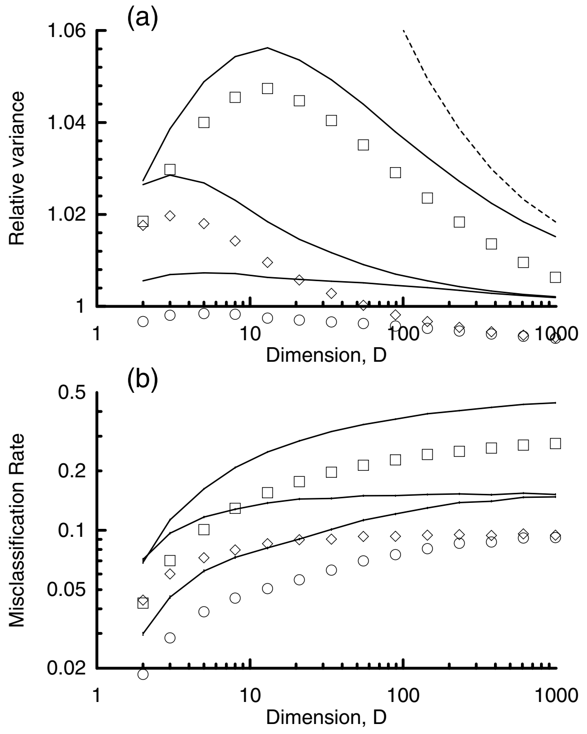

Figure 2: (a) Relative variance for $K = 2$ classes, plotted against the dimension D of the sphere onto which a triplet of points have been randomly placed. Diamonds represent the Manhattan metric, squares represent the Max metric, and circles correspond to the linear combination with $\alpha = 0.25$. The dotted line corresponds to the relative variance achieved by random classification of the data. (b) Probability of misclassification for the same parameters as in (a). Random classification would produce a misclassification rate of 0.5.

As expected, the linear combination performs better than the Manhattan distance. The improvement is substantial for smaller dimensions ($D < 10$), but the difference becomes small for large dimensions. Due to the extra cost that would be incurred in order to implement the linear combination, this alternative was not investigated further.

The data dependent experiment was performed on AVIRIS data sets. Each AVIRIS image was classified with each of the distance measures, so there were twenty trials (5 images × 4 distance measures). Each trial used $K = 16$ clusters and stopped after convergence or fifty iterations. Results are shown in Table 1.

Image	Within-class Variance			Relative increase	
	Euclidean	Manhattan	Max	Manhattan	Max
f960323t01p02_r04_sc01.c.img	3.7533e+06	5.1121e+06*	3.9328e+06	0.3620	0.0478
f970410t01p02_r02_sc02.c.img	3.7806e+06	4.1296e+06	4.3915e+06	0.0923	0.1616
f970620t01p02_r03_sc02.c.img	1.0339e+07	1.1329e+07*	1.1089e+07	0.0958	0.0725
f970701t01p02_r07_sc01.c.img	9.4839e+06	9.9656e+06*	1.0726e+07*	0.0508	0.1309
f970801t01p02_r01_sc01.c.img	3.5210e+06	3.5597e+06	3.5828e+06*	0.0110	0.0175

Table 1: Comparisons of within-class variance for clusterings of AVIRIS data using different variants of k-means. $K = 16$ clusters were used, and the algorithm was run for 50 iterations. An asterisk indicates that convergence was achieved before 50 iterations.

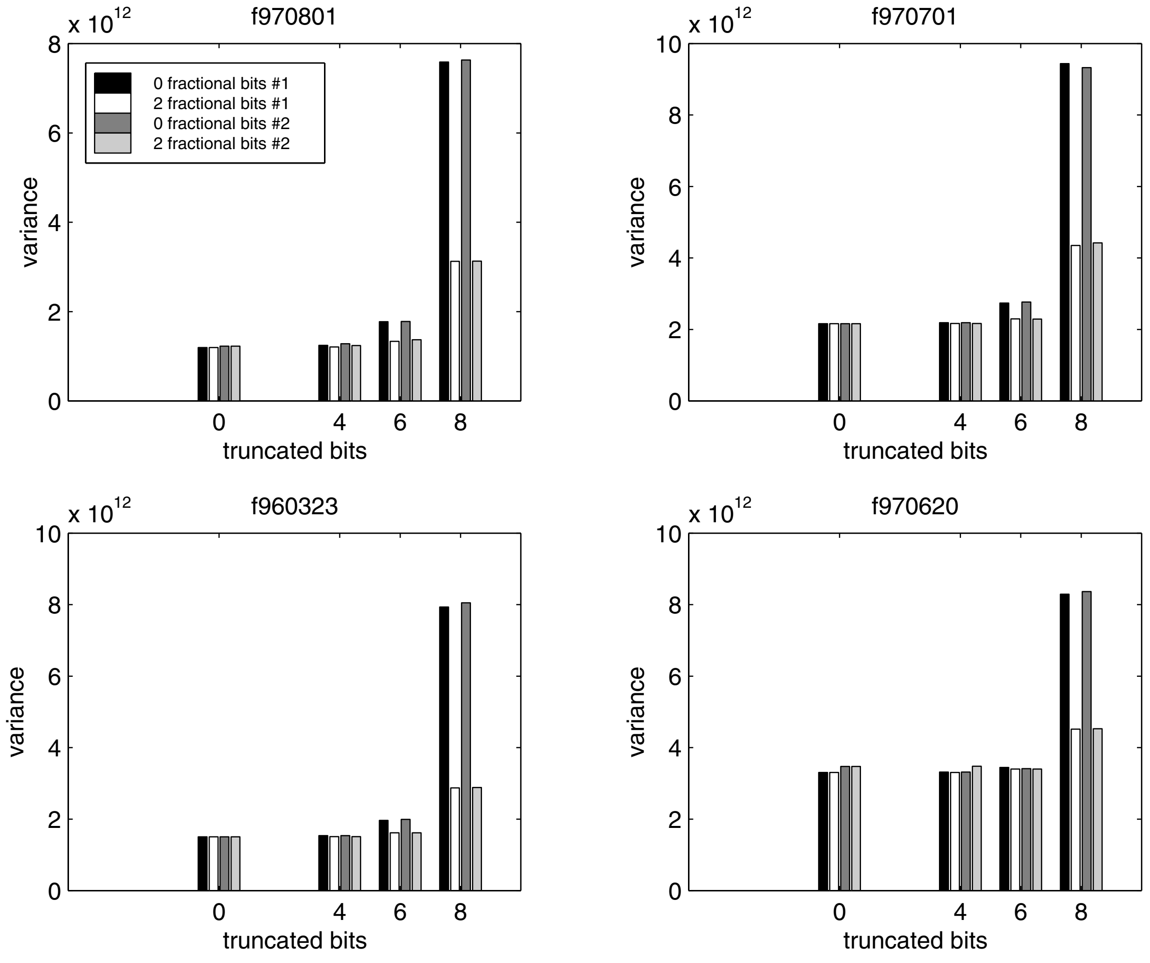

Figure 3: Total within-class variance for AVIRIS data cubes. Four different AVIRIS images were used. Each graph shows the total variances for one image. 16 clusterings were run on each image and total variance was recorded for each. There were two sets of experiments per image, each corresponding to a different initialization. For each initialization, 8 clusterings were run. Four different bitwidths of the input data were used, with 0, 4, 6 and 8 bits truncated, respectively. For each truncation, two cluster center sizes were used; one with the center the same bitwidth as the truncated pixel, the second with the center containing two bits more than the truncated pixel.

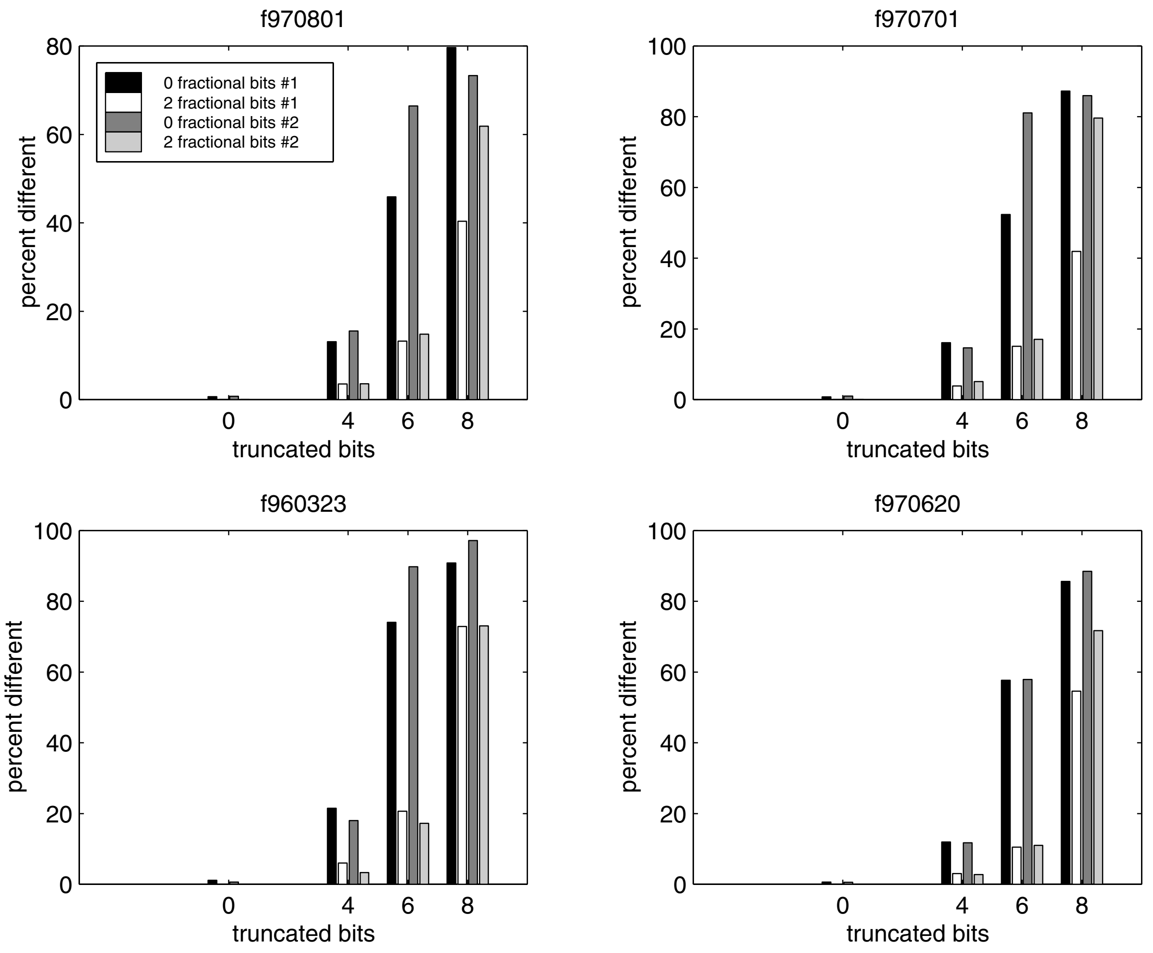

Figure 4: Different classification rates for AVIRIS data cubes. These show results for the same four images and the same 16 experiments per image as the variances. A pixel is classified as different if it ends up in a different class than the case of 0 bits truncated, 2 extra bits used for centers.

Both experiments showed that the Manhattan metric is better than the Max metric for k-means. In the data dependent experiment with AVIRIS data, we found that clusters produced using the Manhattan distance metric were slightly less compact than the clusters produced by the Euclidean metric. The total within-class variance, for the AVIRIS data cubes we measured, increased from 2% to slightly over 30% for Manhattan vs. Euclidean distance, with most of the images exhibiting less than a 6% increase.

3.2 Input Data Truncation

Implementing the Manhattan distance results in significant savings in datapath area, and thus allows for more parallelism in a reconfigurable implementation. However, given the size of the dataset being investigated, more drastic measures were considered to further reduce the size of the calculation of each pixel.

Recall that an AVIRIS data cube contains pixels with 224 channels, 16 bits per channel. Data cubes in this format are obtained from JPL [8]. The actual data, however, is in the range of 12 bits per channel. Since there are so many bits per pixel (12×224 when considering the true dynamic range), the hardware saved by reducing the number of bits used per channel, even by a small amount, is potentially dramatic.

As with the distance metric experiments, we performed both data independent and data dependent experiments to determine the effects of truncating the image data and the cluster centers [10].

A data independent Monte-Carlo trial is defined as follows. First, $K + 1$ points are placed at random on the surface of a D dimensional sphere with unit radius. One of the points is treated as the data point that is being classified, and the other K points are treated as cluster centers. The center to which the data point is closest is identified as the "true classification" for that data point. The data and centers are then truncated to a precision p, and again the (truncated) center which is closest to the (truncated) data point is identified. If the closest center found with the data truncated differs from the "true" center, then a misclassification is recorded. The "excess in-class variance" is also computed – this is the difference of two squared distances: the distance between the data point and the misidentified center, and the distance between the data point and the correctly identified center, with both distances computed using the full precision (non-truncated) data. The excess variance is normalized by dividing by the average (over many trials) of the squared distance of a point to its nearest center.

For these data independent experiments, all data values range from zero to one due to the fact that all points are placed on a D dimensional sphere with unit radius. The precision p is therefore less than one for these experiments ($p = 1$ would correspond to no information at all in the data; while $p = 0.25$ for instance would correspond to data truncated to two bits of precision after the decimal point).

The experiment was repeated for different values of D, K and p. Results show that misclassification rate scales linearly with p, and excess variance scales like p^2. Both measures also increase with larger dimension D, with a scaling that appears to grow as $\sqrt{D}$; this factor can be explained by the fact that the data is normalized to a unit sphere: $\langle x_i^2 \rangle = 1/D$ since $x_1^2 + \ldots + x_D^2 = 1$. Further numerical results (not shown) suggest that these error measures increase roughly with $\sqrt{K}$.

On the whole, these results suggest that we can get good quality classification with fairly severe truncation of the input data.

Since we are interested in applying bit truncation to hyperspectral image sets, we also did experiments with AVIRIS data cubes and measured the effect. Here a reference classification was obtained by using full precision data, and our experiments examined the effect of truncating bits by comparing to that reference classification. We looked at within-class variance – which was measured using the full precision data, even though the clustering was done with the truncated data. Within-class variance is what k-means seeks to minimize, and it is our bottom-line measure of cluster quality. We also counted the number of pixels which were classified differently in the full-precision and the low-precision clusterings. A large difference means that the analyst will be presented with a picture that looks qualitatively different, but it doesn't necessarily mean that the clustering is of lower quality.

The experiment was run on four different AVIRIS cubes. The total within-class variances are shown in Figure 3. For each data cube we varied precision of the data and precisions of the centers. We ran all experiments with two different initializations, since results are sensitive to the the initializations. The same initialization is used for all combination of truncation exercises. Manhattan distance was used as the metric for choosing cluster centers. For these experiments, the difference in pixel classification was also recorded, as shown in Figure 4. In this case, a pixel is recorded as different if the class it is assigned to differs from the the class it would have been assigned to using the full bitwidth of data, full bitwidth of cluster centers, and Manhattan distance. This difference provides a measure of the qualitative differences seen in the results of classification of the images with different amounts of truncation.

Our first observation is that cluster quality is maintained even with considerable truncation. Little variation in total within-class variance is observed, even when only half the input bits ($B = 6$) are used. Another empirical observation is that it makes sense to truncate the data more aggressively than the centers. For instance, it is better to truncate the data to 6 bits but keep the centers at 8 bits than to truncate both to 6 bits. This is good both for reducing memory requirements and processing bandwidth. Using 6 bits per channel reduces the size of the data cube by 50%.

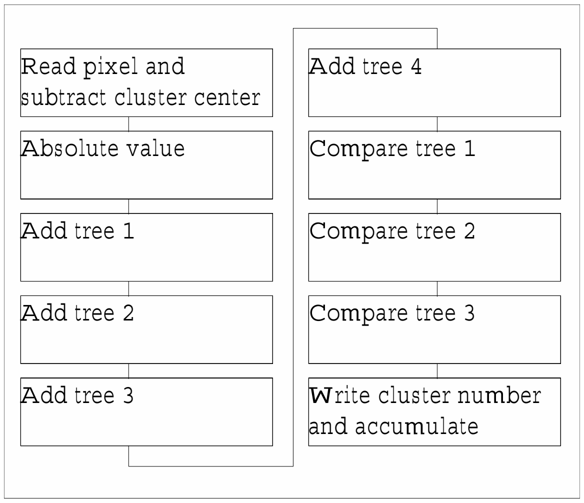

Figure 5: Stages of pixel processing pipeline

4. HARDWARE IMPLEMENTATION OF K-MEANS

We implemented the k-means algorithm on the Annapolis Microsystems Wildstar PCI board with three Xilinx Virtex 1000 FPGAs and 40MB of ZBT SRAM. The Wildstar is in a 500MHz Pentium III workstation. The implementation was written in VHDL and synthesized with Synplicity's Synplify and Xilinx Alliance tools.

The design classifies 614×512 pixel images with 10 channels of 12 bit data per pixel into 8 clusters. (Note that these are simulated MTI data sets.) The design returns the cluster number for each pixel, as well as the accumulated values for each channel of each cluster and the number of pixels in each cluster. The Manhattan distance measure is used to compare pixels to cluster centers. There is no truncation of bits in this implementation.

The FPGAs on the Wildstar are called Processing Elements 0, 1, and 2 (PE0, PE1, PE2). Only PE1 is currently used by the design. PE1 has an interface to the host PCI bus, two 32 bit memory interfaces, and two 64 bit memory interfaces. Registers mapped onto the host PCI bus hold the control signals, cluster centers, and cluster accumulators. The two 64 bit memory ports hold the pixel data, so that 128 bits of the image can be accessed each clock cycle. Each pixel is 120 bits (10 channels by 12 bits), and the image is mapped into the memories so that one whole pixel is accessed each clock cycle with 8 unused bits. One of the 32 bit memory ports is used to hold the cluster number for each pixel, and the other 32 bit memory port is not used. One pixel is classified every clock cycle. The design uses a 10 stage pipeline as shown in Figure 5.

The first stage takes the pixel data and for each channel subtracts from it each cluster center. The second stage takes the absolute value of all the differences from the first stage. The third through sixth stages comprise an adder tree for each cluster. After the sixth stage is done, the distance calculation from the pixel to each of the eight clusters is finished. The seventh through ninth stage comprise a compare tree that select the cluster with the smallest distance to the pixel. The tenth stage writes the cluster number to

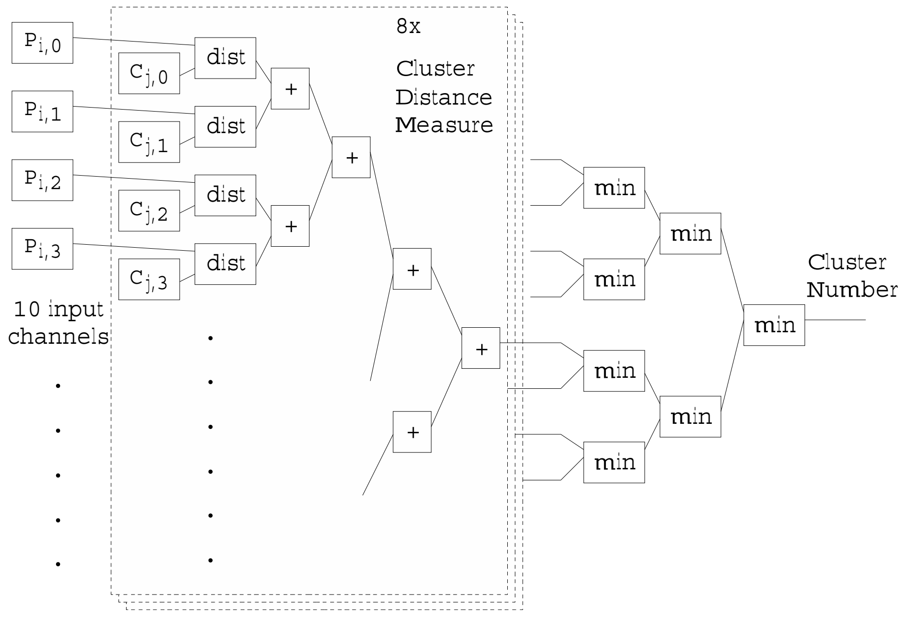

Figure 6: Implementation of a pixel comparison.

memory and accumulates the pixel value into the cluster accumulators.

The k-means algorithm is sensitive to initialization. We use a hierarchical initialization scheme. The host software loads the complete image into the memories of the Wildstar. The k-means algorithm is run in software on a randomly chosen subset of 256 pixels, and the final cluster centers from the subset are used to initialize the cluster centers on the Wildstar. For each iteration, the software writes the cluster centers to the Wildstar, waits for the Wildstar to finish assigning each pixel to a cluster, reads back the accumulated values, and divides the accumulated values for each cluster by the number of pixels in that cluster to create the new cluster centers. One iteration through an image involves a distance computation between 314,368 pixels and 8 clusters each of which have 10 dimensions. Calculations on a single pixel are illustrated in 6. After a complete iteration, 80 division operations are required to compute the new cluster centers. The algorithm converges when the cluster centers are identical for two iterations in a row, signifying that no pixels have changed clusters. After the algorithm converges, the software reads back the cluster number for each pixel and writes an image file with each cluster a different color.

The design was synthesized with a target frequency of 50 MHz. It was successfully mapped at 50 MHz into 75% of a Xilinx Virtex1000. The design was tested with five different 614x512 pixel images. The hardware implementation was approximately 200x faster than the software implementation. One of the images took 65 iterations to complete; the FPGA finished in 0.42 seconds and the software finished in 76.2 seconds. Based on the 50MHz clock frequency and the image size, the time spent by the FPGA processing 65 iterations is 0.41 seconds. The total measured time was 0.42 seconds, including the divisions on the host processor. In other words, the divisions require .01 seconds, or slightly more than 2% of the total processing time.

We plan to incorporate bit truncation into our implementation. Truncation will allow us process more channels of an AVIRIS image. The next goal is to be able to process the full 224 channels of an AVIRIS image by reading in pixels over multiple clock cycles. With simple changes to the host software, we could use two PE's in parallel to process an image. Each PE would classify half of the pixels, and the host would combine the results. The two PE's would be twice as fast as one. With more extensive changes, the third PE on the Wildstar could be used to do the divisions that are done on the host now. With this design, the host can run other tasks while the clustering of images is running.

5. DISCUSSION

Figure 7 shows the results of clustering a multispectral image on the Wildstar board. We have shown that considerable speed up in k-means clustering can be obtained by implementing the clustering on reconfigurable hardware. This speed up is obtained partly by applying transformations that specifically accelerate a hardware implementation of k-means. These transformations result not from peephole optimizations, such as implementing a multiplier with shifts and adds, but rather from global optimizations that involve changes to the algorithm being implemented. The criteria for these optimizations is that they do not unduly compromise the quality of the results.

In the future, as FPGA capacities become larger and experienced hardware designers rarer, the implementation of algorithms in reconfigurable hardware will become more automated. Several researchers are investigating compilers that target reconfigurable computing and start from programming languages similar to C [3, 5] or from Matlab descriptions [1]. The success of such approaches will enable efficient designs to more rapidly be implemented on reconfigurable hardware. They will make our approach both easier to apply and more important. If one were to take a k-means

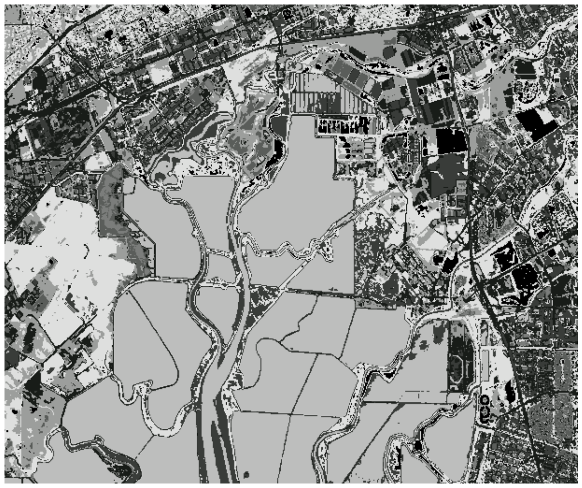

Figure 7: Clustered output of image f970620t01p02_r03_sc02.c.img

algorithm written in C for implementation on a RISC processor and translate it directly to hardware, the resulting design would contain many multiplication operations and either would not fit on the available hardware or would be slow due to the serialization of operations required to share the hardware that can fit. Only by optimizing the algorithm is an efficient implementation realizable. Compilers that generate efficient hardware free the designer of reconfigurable systems to focus on the best algorithm or variant to implement. It is at this level that a designer's attention will have the greatest impact on the implementation.

6. CONCLUSIONS

We have presented the implementation of k-means clustering on an Annapolis Wildstar board that exhibited a speed up of two orders of magnitude over a software implementation. This implementation was only possible as a result of careful analysis of the algorithm and the input data. No special, manual design flow was followed. By considering tradeoffs at the algorithmic level, it is possible to achieve efficiencies in reconfigurable hardware implementations not otherwise possible.

Acknowledgments

We would like to thank Annapolis Microsystems for their support, and Synplicity and Xilinx Corporations for their donations to the Rapid Prototyping Laboratory at Northeastern University. We would like to thank researchers at Los Alamos and at Northeastern University who contributed to this project. This research was funded by a US government agency.

7. REFERENCES

[1] P. Banerjee et al. A MATLAB compiler for distributed, heterogeneous, reconfigurable computing systems. In *IEEE Symposium on FPGAs for Custom Computing Machines*, 2000.

[2] V. Castelli and T. M. Cover. On the exponential value of labeled samples. *Pattern Recognition Lett.*, 16:105–111, 1995.

[3] B. Draper et al. Compiling and optimizing image processing algorithms for FPGAs. In *Workshop on Computer Architecture for Machine Performance*, 2000.

[4] C. Funk, J. Theiler, D. A. Roberts, and C. C. Borel. Clustering to improve matched-filter detection of weak gas plumes in hyperspectral imagery. Technical Report LA-UR 00-3673, Los Alamos National Laboratory, 2000.

[5] M. B. Gokhale, J. M. Stone, J. Arnold, and M. Kalinowski. Streams-oriented FPGA computing in the Streams-C high level language. In *IEEE Symposium on FPGAs for Custom Computing Machines*, 2000.

[6] P.-F. Hsieh and D. Landgrebe. Statistics enhancement in hyperspectral data analysis using spectral-spatial labeling, the EM algorithm, and the leave-one-out covariance estimator. *Proc. SPIE*, 3438:183–190, 1999.

[7] P. M. Kelly and J. M. White. Preprocessing remotely-sensed data for efficient analysis and classification. In U. M. Fayyad and R. Uthurusamy, editors, *Artificial Intelligence 1993: Knowledge-Based Systems in Aerospace and Industry*, volume 1963, pages 24–30, 1993.

[8] NASA, 1999. `http://makalu.jpl.nasa.gov/avaris.html`.

[9] R. A. Schowengerdt. *Techniques for Image Processing and Classification in Remote Sensing.* Academic Press, Orlando, 1983.

[10] J. Theiler, M. Leeser, M. Estlick, and J. J. Szymanski. Design issues for hardware implementation of an algorithm for segmenting hyperspectral imagery. In M. R. Descour and S. S. Shen, editors, *Imaging Spectrometry VI*, volume 4132, 2000.