Effect of Data Truncation in an Implementation of Pixel Clustering on a Custom Computing Machine

Miriam Leeser[a], James Theiler[b], Michael Estlick[a], Natasha Kitaryeva[a] and John J. Szymanski[b]

[a]Department of Electrical and Computer Engineering
Northeastern University, Boston, MA

[b]Space and Remote Sensing Sciences Group
Los Alamos National Laboratory, Los Alamos, NM

ABSTRACT

We investigate the effect of truncating the precision of hyperspectral image data for the purpose of more efficiently segmenting the image using a variant of k-means clustering. We describe the implementation of the algorithm on field-programmable gate array (FPGA) hardware. Truncating the data to only a few bits per pixel in each spectral channel permits a more compact hardware design, enabling greater parallelism, and ultimately a more rapid execution. It also enables the storage of larger images in the onboard memory. In exchange for faster clustering, however, one trades off the quality of the produced segmentation. We find, however, that the clustering algorithm can tolerate considerable data truncation with little degradation in cluster quality. This robustness to truncated data can be extended by computing the cluster centers to a few more bits of precision than the data. Since there are so many more pixels than centers, the more aggressive data truncation leads to significant gains in the number of pixels that can be stored in memory and processed in hardware concurrently.

Keywords: hyperspectral, k-means, image segmentation, image processing, field-programmable gate array (FPGA)

1. INTRODUCTION

There is an increase in multispectral and hyperspectral image data available from a variety of sources, including commercial and government satellites, as well as airborne and ground-based sensors. This is due to an increase in spatial resolution of satellite imagery 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 an unsupervised clustering algorithm to AVIRIS data sets.[1] A single AVIRIS image contains 614×512 pixels with 224 16-bit channels, or approximately 140 MB of 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 per pixel suffice to completely specify the image. The resulting image can be represented by 75K bytes – a compression factor of over three orders of magnitude.

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.[2] 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.[3,4] Schowengerdt[5] 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

For further information, email: mel@ece.neu.edu

filter for each spectral class[6]; 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 (Field Programmable Gate Array) 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, our goals are to simplify the underlying operations as much as possible in order to speed up the calculations and to be able to provide more parallelism. A simpler operation translates to a smaller area datapath which in turn translates to more versions of the datapath 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 a previous paper[7] we investigated the use of alternative distance metrics (such as the Manhattan metric, the Max metric, and a linear combination of the two) which were faster, cheaper, and used less real estate than the Euclidean distance. These metrics did not produce as high quality clusters, but could be implemented with less chip area and narrower datapaths.

In this paper we investigate a more direct form of bitwidth reduction. In order to process an AVIRIS data set with reconfigurable hardware, methods to reduce the amount of data to be handled are required. We conducted a series of experiments to measure the effects of truncating the number of data bits used on the results of the algorithm. The hypothesis being tested is that the result of clustering AVIRIS data with the most significant bits from each channel of each pixel will not differ very much from the results of clustering with the complete data set.

To test this hypothesis, two sets of experiments were devised. In the first set, data-independent experiments were conducted to determine the effect on total within-class variance of truncating the precision of the input bits on the clustering of high dimensional data sets. The second set of experiments tested the same hypothesis using some AVIRIS data cubes.

In the next section we present the k-means algorithm. Next we present results from the data-independent experiments and from the data-dependent experiments using AVIRIS data cubes. In Section 3 we present our current implementation of the k-means algorithm on the Annapolis Wildstar board.

2. K-MEANS: THE ALGORITHM

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 $n = 1, \ldots, N$); 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 this sense of minimum within-class variance) location for each center is the mean of all pixels in each class. And for a fixed choice of centers, the optimal partition assigns each pixel to the cluster whose center is 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 a local minimum. In addition, the final convergence can be very sensitive to the initialization of the algorithm.

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 as:

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

As described in an earlier paper,[7] one can instead employ the Manhattan distance, defined as

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

to assign cluster centers. In simulations with AVIRIS data, we found that clusters produced using this 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.

The advantage of using the Manhattan distance is that it eliminates the need for multiplications, which simplifies the implementation considerably; for AVIRIS data, the distance between one pixel and one center would require $D = 224$ multiplications. In addition the bitwidths required in the datapath are reduced. Euclidean distance requires $2B + log_2 D$ bits in the data path (assuming no square root is implemented) for the total distance, where B is the bits required for the coordinate differences. Using the Manhattan distance, only $B + \log_2 D$ bits are required.

2.1. Bitwidth 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 bit per channel. Data cubes in this format are obtained from JPL.[1] 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. Therefore, we formulated the hypothesis that the quality of the resulting clusters would not be greatly affected if we truncated the number of input bits used, and only used the most significant bits in each band to assign pixels to classes.

In this discussion we use B to denote the number of bits in a data word, and p to denote the precision; p is the difference between adjacent truncated values. For example, for the $B = 12$ bits in an AVIRIS channel, the data values are integers between 0 and 4095, and the precision is $p = 1$. If data are truncated to $B = 8$ bits, then there are only $2^B = 256$ distinct values, but range is still 0 to 4095, so only multiples of $p = 16$ are permitted. In general, for our 12-bit data sets, we have $p = 2^{12-B}$.

If the pixel data is truncated to a precision p and if it is within p of the border that separates two centers, then it is possible that the pixel will be misassigned. This is illustrated in Figure 1 where the two plus signs represent centers and the dark areas the volume of spectral space in which a pixel will be misassigned. This volume is proportional to p; thus the rate of misclassification is expected to scale linearly with p. If the centers themselves are inaccurate, the effect will be to increase the volume of this region of misclassification. The error due to center truncation is also expected to scale linearly with the precision to which the centers are truncated, though not necessarily with the same coefficient.

If a pixel is misassigned due to truncation error, that means the pixel had to be near the border between two centers. It follows that the effect of such a misclassification on the total within-class variance will be relatively small; in fact it will scale linearly with p. Multiplying this error by the misclassification rate, we find that effect of imprecision on total within-class variance scales as p^2. This suggests that we can impose a small truncation error on the data without having a large impact on the within-class variance.

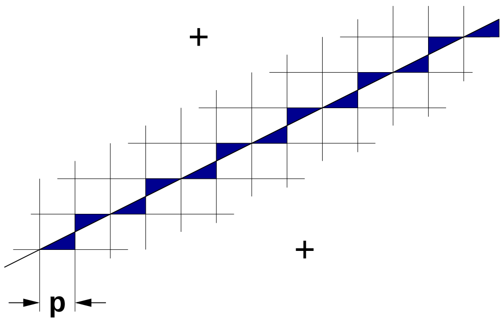

Figure 1. This figure illustrates the region on the border between two centers, represented here as plus signs. Pixel values that are truncated to precision p are effectively moved to the center of the square in which they originally resided. For those points in the shaded region, this movement puts them on the other side of the border which means that they will be misclassified. It is clear that the area of this region of misclassification scales linearly with the precision p.

To test this hypothesis we did two sets of experiments. The first set of experiments tested the hypothesis using an idealized model of D dimensional space. The second set ran experiments on data cubes of AVIRIS data.

Note that our experiments involve truncating the input data, or, in other words, using less than the full dynamic range of the data to calculate the result. Since the number of bits of input data is reduced, the number of bits for all intermediate calculations is also reduced. Clearly the size of the cluster centers can also be truncated. Since each channel gets compared to each cluster center, we experimented with two scenarios:

- Truncate the pixel data to B bits; truncate the centers to the same bitwidth, B.
- Truncate the pixel data to a bitwidth B; truncate the centers to $B + 2$ bits of precision.

2.2. Data Independent Assessment of Truncating Bitwidths

The purpose of the "data independent" assessment is to obtain generic results that do not depend on the details of a particular data set, and to confirm the scaling of misclassification and in-class variance with precision p that was discussed above. These results are based on Monte-Carlo experiments that correspond to a single iteration of the k-means algorithm applied to random data. We have previously describe a similar data-independent experiment[7] for assessing choice of distance metric.

A single 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 for $K = 2$ are shown in Figures 2. It is seen that, as argued above, 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

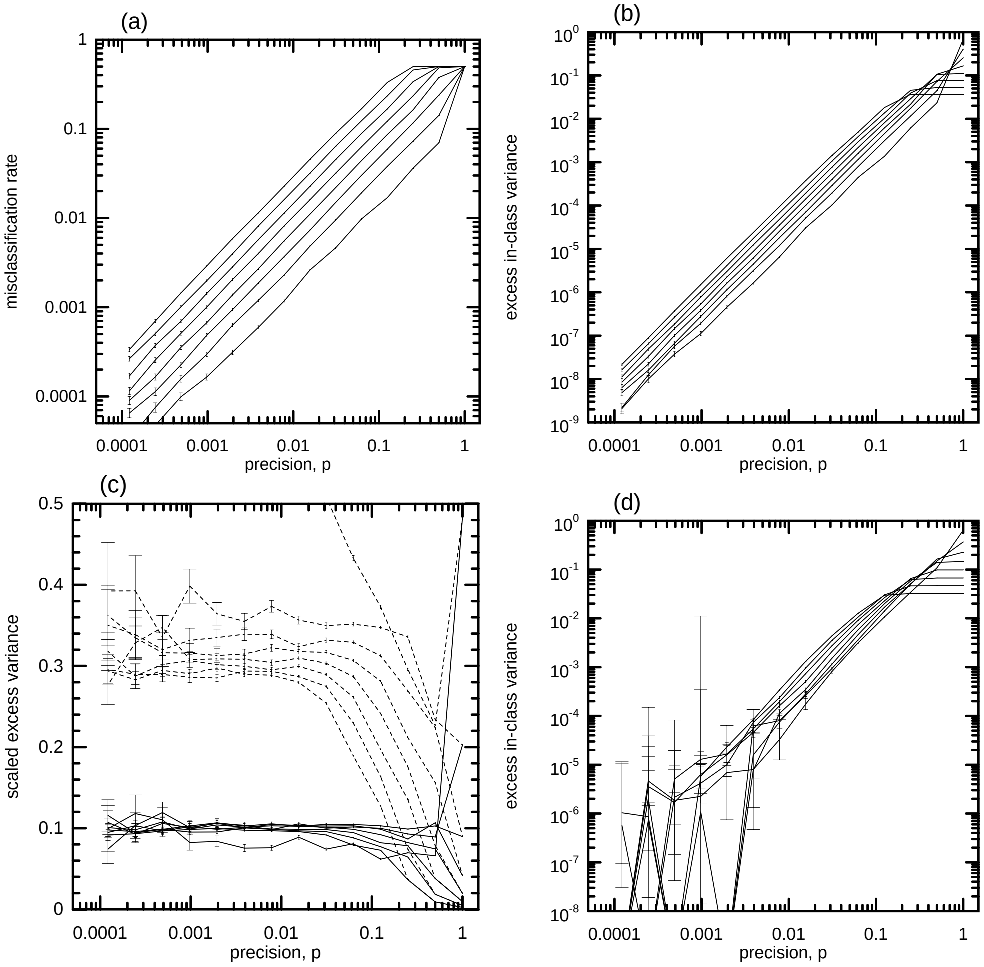

Figure 2. (a) The misclassification rate is plotted against the precision p for dimensions D which are powers of two in the range from 2 to 256. The bottom curve corresponds to $D = 2$ and the top curve to $D = 256$. Here the centers were computed to full precision, and misclassification is seen to increase linearly with p and roughly as the square root of D. **(b)** The cost of misclassification is measured by the change in the in-class variance. This excess in-class variance is seen to scale quadratically with p, and again with $\sqrt{D}$. **(c)** In this panel we plot excess variance divided by $p^2\sqrt{D}$; this corresponds roughly to the coefficient of the observed scaling. The solid lines correspond to the same data as in (b); this illustrates the scaling with p^2 and $\sqrt{D}$. The dashed lines correspond to the values obtained when the centers are truncated to the same precision as the data. **(d)** The previous three panels employed the standard Euclidean distance metric to assign points to centers. If the Manhattan metric is instead employed, the same overall scaling is observed for larger values of p, but for smaller values of p the situation becomes more complicated. In some cases, the errors introduced by the Manhattan metric and by precision truncation appear to cancel.

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.

2.3. Data Dependent Assessment of Truncated Bitwidths

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.

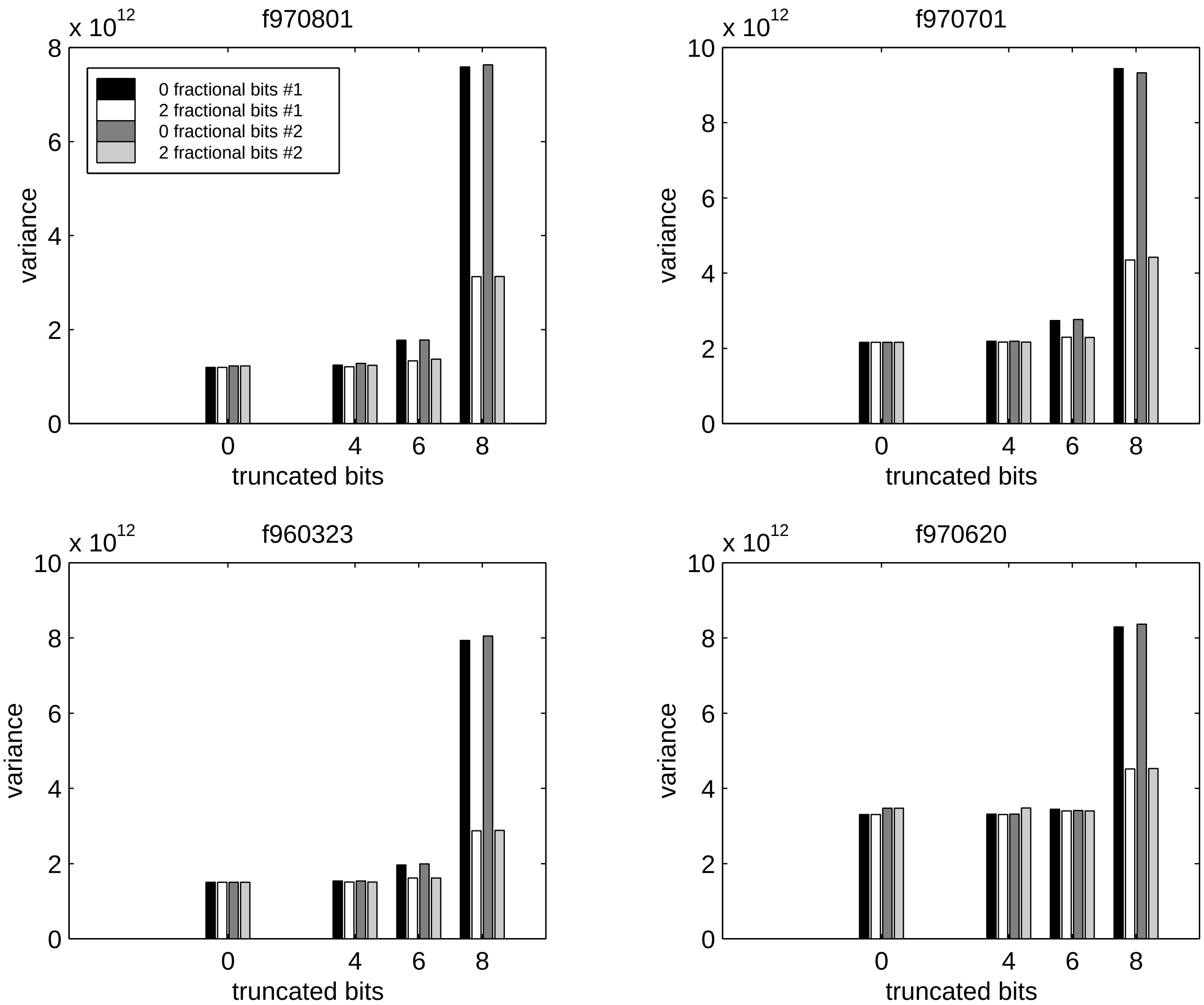

Figure 3. Total within-class variance for AVIRIS data cubes. Four different AVIRIS images were used. Each graph shows the total variances for one image. 16 clusterings were run on each image and total variance was recorded for each. There were two sets of experiments per image, each corresponding to 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.

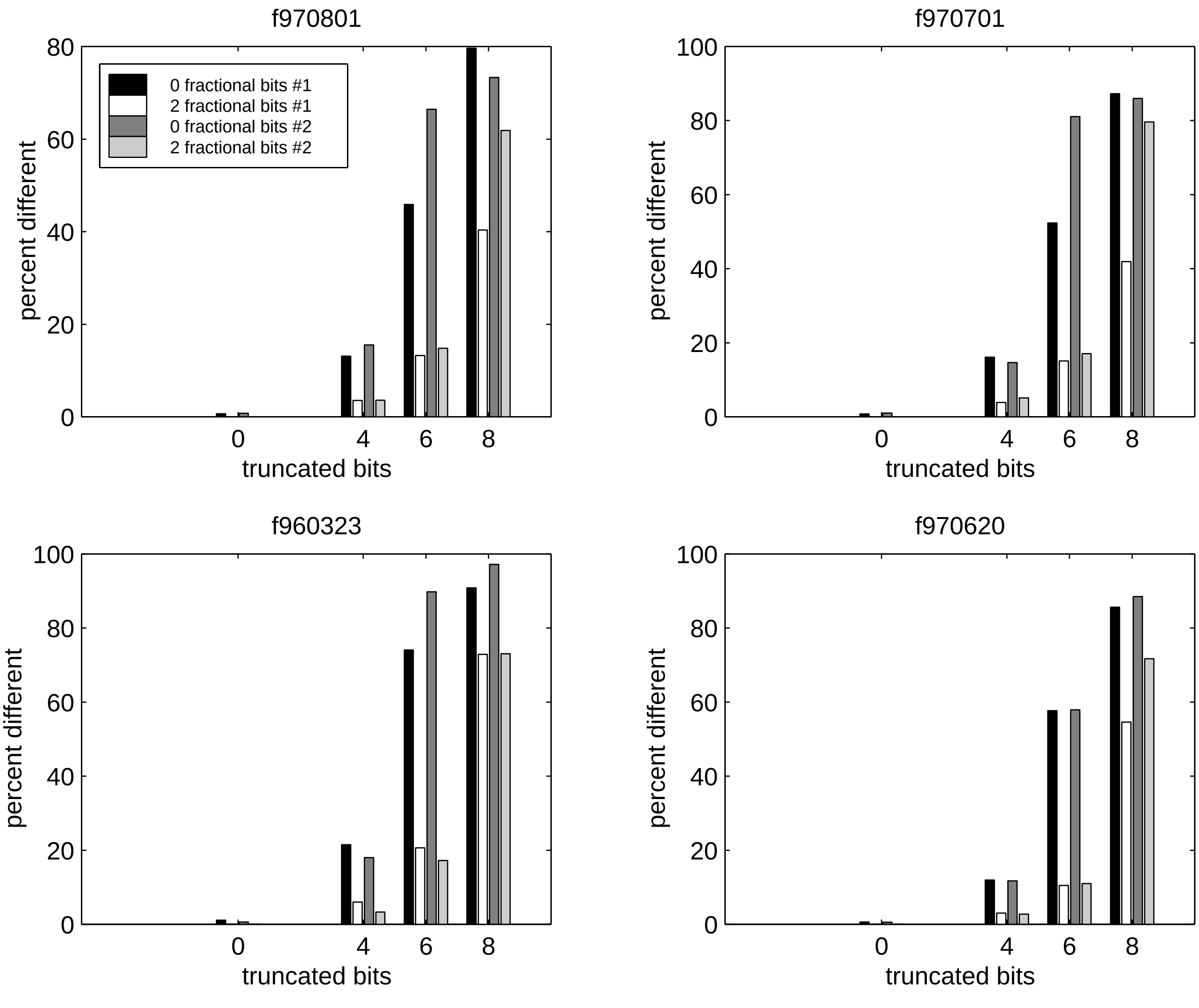

Figure 4. Different classification rates for AVIRIS data cubes. These show results for the same four images and the same 16 experiments per image as the variances. A pixel is classified as different if it ends up in a different class than the case of 0 bits truncated, 2 extra bits used for centers.

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%.

3. HARDWARE IMPLEMENTATION OF K-MEANS

We have implemented k-means clustering on a Wildstar board from Annapolis Microsystems. The particular board, a member of the WS/XCV1000 family, has three Virtex 1000s, each with over 1 Million gate equivalents.[8] These Virtex chips are referred to as processing elements (PEs 0, 1 and 2). In addition 40 Megabytes of onboard RAM is available, which is used for pixel storage in our application.

The k-means algorithm consists of two stages: the first iterates over all the pixels and determines the new cluster number to which each pixel is assigned. Accumulated values for clusters are kept during this stage. The second is run once per image iteration and determines the new cluster center values. We implement the first stage on the Virtex chips and the second on the host. The design works as follows:

1. Calculate initial mean table on host using a small subset of image pixels and a hierarchical initialization methodology.
2. Download the image to memory on the Wildstar board.
3. Download the mean table to PE1.
4. Calculate new cluster numbers for all image pixels, and accumulate statistics.
5. Communicate statistics to host for new mean table calculation.
6. Calculate new mean table on host.
7. If termination condition is not met then go to 3, else done.

There are several possible termination conditions: one may run for a fixed number of iterations; or one may run until the algorithm fully converges, which is detected by the lack of change in center values from one iteration to the next; or one may aim for near-convergence based on a heuristic criterion. For maximum flexibility, the termination is currently determined by the host.

It is important to note that the image data is downloaded once to the Wildstar board and never needs to be uploaded. Due to the large quantity of image data (134 Megabytes for a full data cube), moving it from one location to another is time consuming and is avoided in our implementation. The information that must be uploaded to the host after each iteration is the minimum set required to calculate the new means. This includes the number of pixels assigned to each class and accumulated values for each channel for each class calculated during pixel assignment. The result of clustering is an image with each pixel represented by a pointer to the class to which it is assigned. This image requires 75K bytes for a full AVIRIS data cube, and is uploaded once to the host after all iterations are complete.

Our current, working implementation of k-means works on the full 12 bit data range per channel, but uses a reduced data cube. We cluster 256×256 pixel images, each with 10 channels of 12 bit data, using 8 means. This implementation produces considerable speed up over a host PC running the same algorithm. The Wildstar completes 50 iterations of the clustering algorithm on a dataset in 63 msec running at 40MHz, compared to 14.7 seconds for the same algorithm run on a 500MHz Pentium. The Wildstar implementation runs over 200 times faster than the workstation version.

Extrapolating these results, we expect to be able to fit 20 channels of truncated 6 bit data clustered to 8 means on a Virtex 1000, and plan to handle higher dimensions. Our current implementation is fully pipelined, and additional area can be saved by doing some operations sequentially rather than in parallel. Recently announced Virtex chips, such as the Virtex 3200 from Xilinx, will allow us to triple the number of channels of data that can be handled.[8]

Our current implementation of k-means uses only a single Virtex chip (PE1) to do the actual pixel clustering. Another chip (PE0) is dedicated to interfacing with the host and downloading the image to the memory on the Wildstar board. The third chip (PE2) is currently unused, but could be used either to do pixel clustering on a different part of the data cube and thus double the parallelism of the first stage of the implementation, or to do the mean calculation that is currently done by the host. Both these scenarios are being investigated.

4. DISCUSSION

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 peep-hole 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.

We employ a standard design flow for programming the Wildstar board. We describe the hardware in VHDL, and used Synplicity design tools[9] and Xilinx place and route tools[10] to generate the final design. Currently, more efficient designs can be obtained by manually designing the datapath. For example, in a companion project at Los Alamos,[11] a very efficient implementation of the PPI (pixel purity index) algorithm on reconfigurable hardware was obtained through manual design of the datapath.

In the future, as FPGA capacities become larger and experienced hardware designers rarer, such manual approaches will be employed less and less. Several researchers are investigating compilers that target reconfigurable computing and start from programming languages similar to C[12,13] or from Matlab descriptions.[14] 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 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 that will make best use the available resources.

In this paper, we focus on data reduction by truncating the input data bitwidths. Tools that determine the bitwidths required for intermediate calculation will allow more dense implementation of algorithms. For example, the Bitwise tool[15] determines the bits needed for intermediate calculations assuming precise bitwidths of operators were not specified by the designer. This and similar approaches free the designer from specifying details of the implementation without sacrificing efficient hardware implementation. They complement our approach, which involves examining the data and determining how much of the input data set is significant with respect to affecting the results of the algorithm being applied.

Our immediate goal is to apply the results of our data truncation experiments to the k-means implementation to segment larger data sets in hardware. We expect to be able to handle full AVIRIS data cubes using a combination of input data truncation and more serialization of the hardware implementation. Since our current design provides a speed up factor of 200 when implementing the design in hardware, some serialization can be implemented while still obtaining significant speed up in hardware.

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.

REFERENCES

1. NASA, 1999. `http://makalu.jpl.nasa.gov/avaris.html`.
2. P. M. Kelly and J. M. White, "Preprocessing remotely-sensed data for efficient analysis and classification," in *Artificial Intelligence 1993: Knowledge-Based Systems in Aerospace and Industry*, U. M. Fayyad and R. Uthurusamy, eds., *Proc. SPIE* **1963**, pp. 24–30, 1993.
3. V. Castelli and T. M. Cover, "On the exponential value of labeled samples," *Pattern Recognition Lett.* **16**, pp. 105–111, 1995.
4. 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**, pp. 183–190, 1999.
5. R. A. Schowengerdt, *Techniques for Image Processing and Classification in Remote Sensing*, Academic Press, Orlando, 1983.
6. C. Funk, J. Theiler, D. A. Roberts, and C. C. Borel, "Clustering to improve matched-filter detection of weak gas plumes in hyperspectral imagery," Tech. Rep. LA-UR 00-3673, Los Alamos National Laboratory, 2000.
7. J. Theiler, M. Leeser, M. Estlick, and J. J. Szymanski, "Design issues for hardware implementation of an algorithm for segmenting hyperspectral imagery," in *Imaging Spectrometry VI*, M. R. Descour and S. S. Shen, eds., *Proc. SPIE* **4132**, 2000.
8. Xilinx Corporation, 2000. `http://www.xilinx.com/products/virtex.htm`.
9. Synplicity Corporation, 2000. `http://www.synplicity.com`.
10. Xilinx Corporation, 2000. `http://www.xilinx.com/products/software/software.htm`.
11. D. D. Lavenier, J. Theiler, J. J. Szymanski, M. Gokhale, and J. R. Frigo, "Fpga implementation of the pixel purity index algorithm," in *Reconfigurable Technology II: FPGAs and Reconfigurable Processors for Computing and Applications*, J. Schewel, P. M. Athanas, C. H. Dick, and J. T. McHenry, eds., *Proc. SPIE* **4212**, 2000.
12. B. Draper *et al.*, "Compiling and optimizing image processing algorithms for FPGAs," in *Workshop on Computer Architecture for Machine Performance*, 2000.
13. 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.
14. P. Banerjee *et al.*, "A MATLAB compiler for distributed, hetergeneous, reconfigurable computing systems," in *IEEE Symposium on FPGAs for Custom Computing Machines*, 2000.
15. M. Stephenson, J. Babb, and S. Amarasinghe, "Bitwidth analysis with application to silicon compilation," in *Proceedings of the SIGPLAN conference on Programming Language Design and Implementation*, June 2000.