Co-design of Software and Hardware to Implement Remote Sensing Algorithms

James Theiler, Jan Frigo, Maya Gokhale, and John J. Szymanski

Space and Remote Sensing Sciences, Los Alamos National Laboratory, Los Alamos, NM 87545

ABSTRACT

Both for offline searches through large data archives and for onboard computation at the sensor head, there is a growing need for ever-more rapid processing of remote sensing data. For many algorithms of use in remote sensing, the bulk of the processing takes place in an "inner loop" with a large number of simple operations. For these algorithms, dramatic speedups can often be obtained with specialized hardware. The difficulty and expense of digital design continues to limit applicability of this approach, but the development of new design tools is making this approach more feasible, and some notable successes have been reported. On the other hand, it is often the case that processing can also be accelerated by adopting a more sophisticated algorithm design. Unfortunately, a more sophisticated algorithm is much harder to implement in hardware, so these approaches are often at odds with each other. With careful planning, however, it is sometimes possible to combine software and hardware design in such a way that each complements the other, and the final implementation achieves speedup that would not have been possible with a hardware-only or a software-only solution. We will in particular discuss the co-design of software and hardware to achieve substantial speedup of algorithms for multispectral image segmentation and for endmember identification.

Keywords: co-design, configurable computing, field-programmable gate array (FPGA), k-means, endmembers

1. INTRODUCTION

Studying algorithms requires thinking in several ways:
creatively, to discover an idea that will solve a problem;
logically, to analyze its correctness;
mathematically, to analyze its performance; and
painstakingly, to express the idea as a detailed
sequence of steps so it can become software.

– Christopher Van Wyk[1]

An algorithm is a procedure for getting something done. The specification of an algorithm must be detailed and precise; it is an exact and painstaking science. But the design of algorithms is an art. The recent emergence of reconfigurable hardware, and in particular the commercial availability of Field Programmable Gate Array (FPGA) chips and boards,[2–4] has added a new dimension to the design space. By exploiting the inherent fine-grained parallelism available in programmable hardware, one can obtain speedups of one to two orders of magnitude over what is achievable on a general purpose serial processor – and this is in spite of the fact that the FPGA clock speed is typically an order of magnitude slower than the microprocessor.[5] Unfortunately, the price paid for this higher performance is a much longer design cycle. There is currently a vigorous research and development effort to create design tools for reducing the length of this cycle,[6–8] but the practical upshot for scientific research (where one, or at most a few, of the implementations will be deployed) is that it is not cost-effective to design a full-up hardware implementation for any but the simplest algorithms.

On the other hand, even simple algorithms have their place. The lowly dot product is an example that is as widely applicable as it is mathematically trivial. An efficient dot product can be implemented on an FPGA,[9] but the dot product is also a good example of an "algorithm" which is usually just one part of the real algorithm (eg, an adaptive matched filter[10,11]) that you may want to implement for your remote sensing data analysis.

E-mail: {jt,jfrigo,maya,szymanski}@lanl.gov

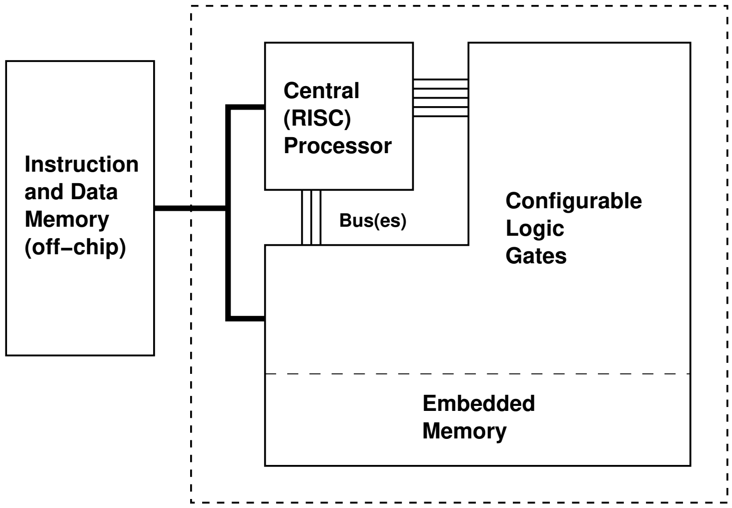

Figure 1. Typical Configurable System on a Chip (CSoC) architecture. A central RISC processor is built into the same chip as the logic gates and embedded memory modules, and both have access to local on-board, but off-chip, memory. The size and number of buses connecting the central processor to the configurable logic can itself be configured to achieve a bandwidth that is appropriate for the application.

Hardware implementation is sometimes seen as a brute force approach to speeding up algorithms. Generally, to implement an algorithm in hardware requires a lot more effort than to implement it in software. And generally, to fit an algorithm into hardware, certain approximations, truncations, and/or simplifications must be made. These approximations must be traded against the raw speed provided by the fine-grained parallelism in a hardware implementation.

One very nice example of a co-design was developed by Porter *et al.*[12,13] It employs a genetic algorithm in software that specifies the settings of hardware registers to direct a generic image processing operator that is implemented in an FPGA. Compared to the original all-software version of this algorithm,[14] this hardware-accelerated system has a much smaller operator set, but it makes up for this with a search through operator space that is roughly two orders of magnitude faster. This example also illustrates the need to decompose an algorithm so that there is a simpler part (often an "inner loop") that can be sped up in hardware in such a way that doesn't interfere with potentially complicated branching logic (in the "outer loop"). This complicated outer loop logic is what allows you to design sophistication into your algorithm, sophistication that may enable further speedup, or that may simply be necessary to achieve a sophisticated goal.

For algorithms that employ substantial software and hardware components, the biggest challenge is the communication bandwidth between them. One answer to this bottleneck is a hybrid processor that combines a traditional microprocessor and programmable logic on the same chip.[15–17] (see Fig. 1.) Such hardware is just now becoming commercially available.[18–27] We have recently used the Altera Excalibur board with a soft-core 32-bit NIOS RISC processor[18] as part of a co-design for k-means clustering.[28] Unfortunately, the promise of seamless integration of a fast central processor surrounded by acres of programmable logic remains to be fulfilled. We achieved a 15% speedup, but were limited by slow memory access and a slow central processor – since the processor on the Excalibur is implemented as a "soft core" it is constrained to run at the same clock speed as the programmable logic itself. With a faster hybrid processor (eg, 10x faster than configurable circuits) or dual port memories that both processor and configurable circuits can access, then we would expect 10x speedup over sequential algorithms. We furthermore extrapolated the expected performance to the Xilinx-PowerPC hybrid[19] and predicted the possibility for two orders of magnitude acceleration.

In the following two sections, we will discuss the possibilities for further co-design opportunities – these simultaneously employ "smart" software and "brute force" hardware to speedup two problems of interest in remote sensing: image segmentation and endmember estimation.

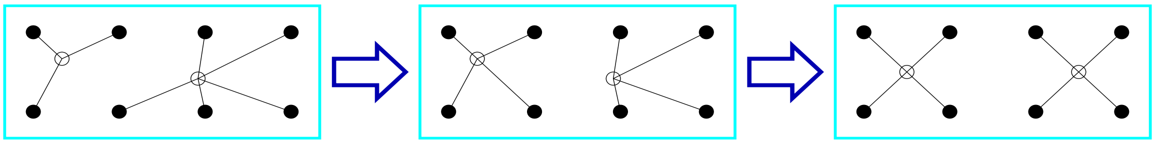

Figure 2. The idea of the k-means algorithm. Data points are represented as filled circles, and centers are open circles. The first step assigns each data point to the center that it is closest to. The second step (re)locates the centers to be the mean value of all the points in the cluster.

2. CLUSTERING AND SEGMENTATION

Although the first published references to the k-means algorithm can be traced to the mid-1960's,[29–32] the basic idea for the algorithm is quite simple (see Fig. 2) and is sometimes called the "generalized Lloyd algorithm" since it is a vector version of Lloyd's original scalar quantization algorithm which was developed in the late-1950's.[33] The fine-grained parallelizability of the algorithm has long been noted,[34] and this makes it a particularly attractive target for hardware implementation. The k-means algorithm is a kind of special case of the Expectation Maximum (EM) algorithm introduced and analyzed by Dempster *et al.*[35] It can be proved to result in better approximations to a local optimum with every iteration; a global optimum, unfortunately, is not guaranteed. The literature on clustering algorithms is quite vast, and there are many reviews[36–41]; partly this is because there is such a wide range of applications.

While multi- and hyper-spectral imagery provide unprecedented amounts of information about a scene, it also presents the image analyst with something of a headache. It is not humanly possible to look at all those channels at the same time. By clustering the data, one provides not only a substantial (albeit lossy) compression, but also a "picture" in which pixels with similar spectral signature are identified by a unique (and usually false) color.

As well as reducing the data for quicklook views, clustering also provides an organization of the data that can be useful for other processing downstream. For example, Kelly and White[42] employ clustering as a preprocessing step to speed up interactive/exploratory manipulations of large data sets. Also, Schowengerdt[43] 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. Several authors have shown that clustering the data beforehand improves the performance of algorithms which attempt to "learn" features from a small number of labelled examples.[44–49] The use of clustering for image restoration has also been explored in some detail; see Sheppard *et al.*[50] for a recent review. For remote detection and characterization of gaseous plumes, the ground scene ceases to be the signal of interest, and becomes instead the clutter. Clustering provides a way to reduce this background clutter, because the within-class variance of a segmented image can be made much smaller than the overall variance of the image as a whole.[51] The issues of clustering and pixel mixing are somewhat at odds with each other, but a paper by Stocker and Schaum[52] points to one approach for combining them.

In the subsections that follow, we will describe both previous and ongoing work to modify the traditional k-means algorithm so as to speed it up with the aid of programmable logic hardware.

2.1. Low-level variants of k-means

One of the first, and most straightforward, ways to alter a floating-point serial-processor algorithm is to truncate the precision of the data and/or of the intermediate computations, and to do the computation in fixed-point arithmetic. We found in Leeser *et al.*[53] that good clusterings could be achieved, even with considerable truncation of the data. We found that it was advantageous to allocate two more bits of precision to the centers than to the data, but this is relatively cheap since there are few centers and many data points.

Truncating the precision of the data still keeps the algorithm more or less intact; a less obvious way to alter the algorithm is to incorporate approximations which are more amenable to hardware. At the core (and in the inner loop) of the k-means algorithm is the computation of distance between the centers and the data points. By altering the distance metric itself, we were able to achieve a hardware implemention which greatly enhanced speedup.

In one dimension, distance between two points is straightforwardly given by the absolute difference between their coordinate values. In higher dimension, the distance between two points can be expressed in terms of the

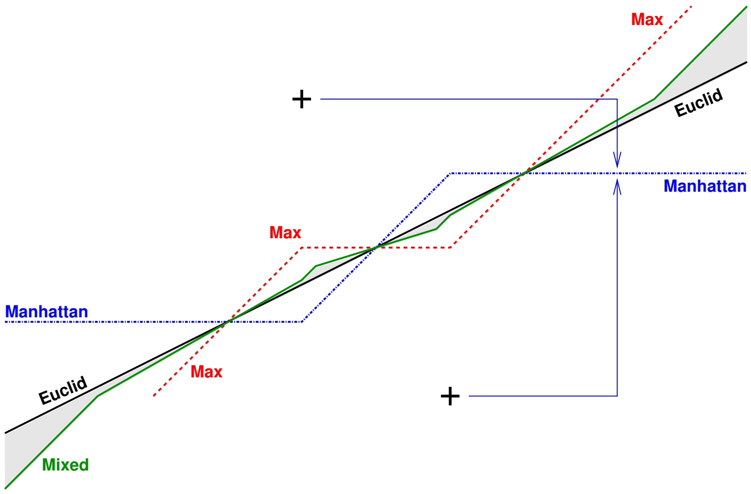

Figure 3. The boundary between classes with centers indicated by crosses in the above figure depends on the distance metric. For instance, the two arrows leading from the class centers to the Manhattan boundary are both the same length (in the Manhattan sense). The jagged line that roughly follows the "correct" Euclidean boundary is produced by the mixed distance measure with $\alpha = 0.25$. Points which fall inside the shaded area between the mixed and the Euclidean boundary would be misclassified by the mixed distance metric. Note how much more accurately the mixed boundary (compared to the Max or Manhattan boundaries) approximates the true Euclidean boundary.

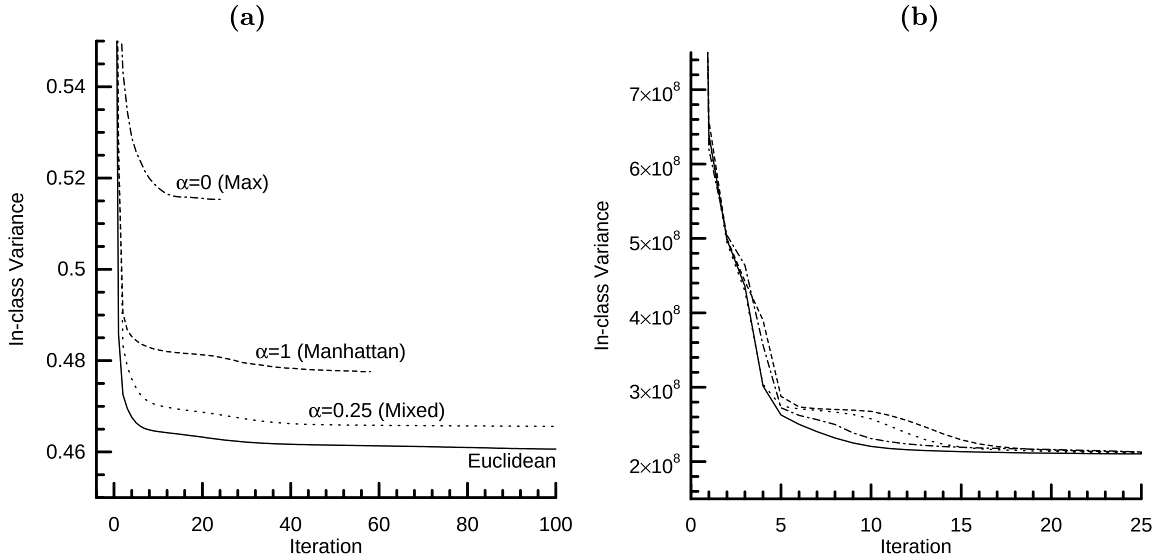

Figure 4. Comparison of k-means performance for different distance metrics. **(a)** This is for $N = 10^5$ points randomly distributed in a $D = 8$ dimensional unit cube, and clustered into $K = 8$ classes. It is evident that the in-class variance is minimized if the "correct" (but expensive) Euclidean distance is used. The Max and Manhattan metric are two alternatives which are much cheaper to implement in hardware, but which do not perform as well as the Euclidean. The Mixed metric is a linear combination of the Max and Manhattan metric; this is still cheaper than the Euclidean metric, but performs nearly as well. **(b)** A multispectral image was created from an AVIRIS hyperspectral datacube[54] (flight number f960323t01p02_r04_sc01) resampled to produce $D = 10$ channels corresponding to bands available on the MTI satellite.[55] This $N = 512 \times 614$ image was clustered into $K = 8$ classes. The theoretical differences between the different distance metrics, seen in panel (a), are much less evident in real data.

one-dimensional coordinate-wise distances. A general family of such distances is parameterized by p:

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

In this family, $p = 2$ corresponds to Euclidean distance, which is a rotationally-invariant measure which furthermore corresponds directly to the in-clsss variance which the k-means algorithm attempts to minimize. Although $p = 2$ is theoretically optimal, two other members of this family, $p = 1$ and $p \to \infty$, are particularly cheap to compute. The Manhattan distance ($p = 1$) is the sum of absolute values of coordinate differences, while the Max distance ($p \to \infty$) is simpy the maximum of all the absolute values of the coordinate differences. Since the error made by the Max distance is usually in the opposite direction as the error made by the Manhattan distance (see Fig. 3), we introduced a better approximation[56] which can be achieved with a simple linear combination of these two:

$$\|\mathbf{x} - \mathbf{c}\|_\alpha = \alpha\, d_{\text{Manhattan}}(\mathbf{x}, \mathbf{c}) \;+\; (1 - \alpha)\, d_{\text{Max}}(\mathbf{x}, \mathbf{c}) \tag{2}$$

We found that $\alpha = 0.25$ was near the optimal value over a wide range of dimensions and furthermore, being a (negative) power of two, was advantageous for hardware implementation. The relative performance of the different distance metrics is shown in Fig. 4, and discussed in more detail elsewhere.[56–58]

2.2. Block updates of centers

In the classic k-means algorithm one sweeps through all the data, assigning each data point to the center it is closest to. At the end of this sweep, the centers are recomputed by taking the mean value of all the points with the label corresponding to that center. However, faster convergence is generally observed when the centers are updated on the fly; that is, centers are recomputed after each new (re)assignment. This is sometimes called the "continuous k-means" algorithm,[60] but the basic idea has been around for a long time.[39] Fig. 5 compares the convergence rate for the standard ($B = N$) and on-the-fly ($B = 1$) versions of the k-means algorithm for some example data.

Unfortunately, updating the centers on the fly is hard to do in hardware; it requires (among other things) a division operation for each data point, instead of just one for each center at the end of the sweep through all the data points. In designs by Leeser *et al.*[56–59] the division operation was offloaded to the general-purpose processor. But it would not be feasible to do this every time a data point was reassigned.

The on-the-fly approach also de-parallelizes the algorithm. In the standard approach, the assignment of a point to a class depends only on fixed centers (so all points could be assigned independently – and in parallel if desired). But the update of centers every time a data point is reassigned will affect subsequent class assignments. A branch condition has been introduced into the "tight inner loop" and the hardware approach is not able to take advantage of this software speedup.

To achieve the software speedup provided by dynamically updated centers, while still using hardware for the main inner-loop acceleration, we can employ a compromise in which class assignments are performed on blocks of B pixels at a time. The centers are only updated between blocks. We see in the experiments in Fig. 5 (on both synthetic and real data) that as long as $B \ll N$, the improvement obtained by on-the-fly updates will still be obtained for relatively large values of B. Fig. 6 shows the dataflow for the implementation reported by Gokhale *et al.*[28] for k-means with blocks of B assignments at a time.

Using this implementation, we performed an experiment to determine the effect of block size B on the overall speed of the k-means co-design. We know that smaller block size leads to faster convergence in software (see Fig. 5), but we expect that larger block sizes will be more efficient in hardware, so we were interested in finding an optimum.

We used random data with $D = 8$ spectral bands, and $b = 8$ bits per point, and we clustered into $K = 8$ classes. We evaluated the algorithm by increasing the size of B, but we found that for $B > 8$, the execution speedup was essentially independent of B, and essentially the same value that we reported earlier,[28] about fifteen percent – not enough yet to be practical.

The bottleneck in this computation is the bandwidth between the central processor and the configurable logic. Using the parallel I/O or memory mapped standard interfaces on the Excalibur Board, we found that it takes 11 cycles to send one 32-bit word.[28] This is a fundamental limitation of the RISC processor (reduced instruction set means more instruction cycles!) being soft-coded into the configurable logic, and therefore running at the same rate

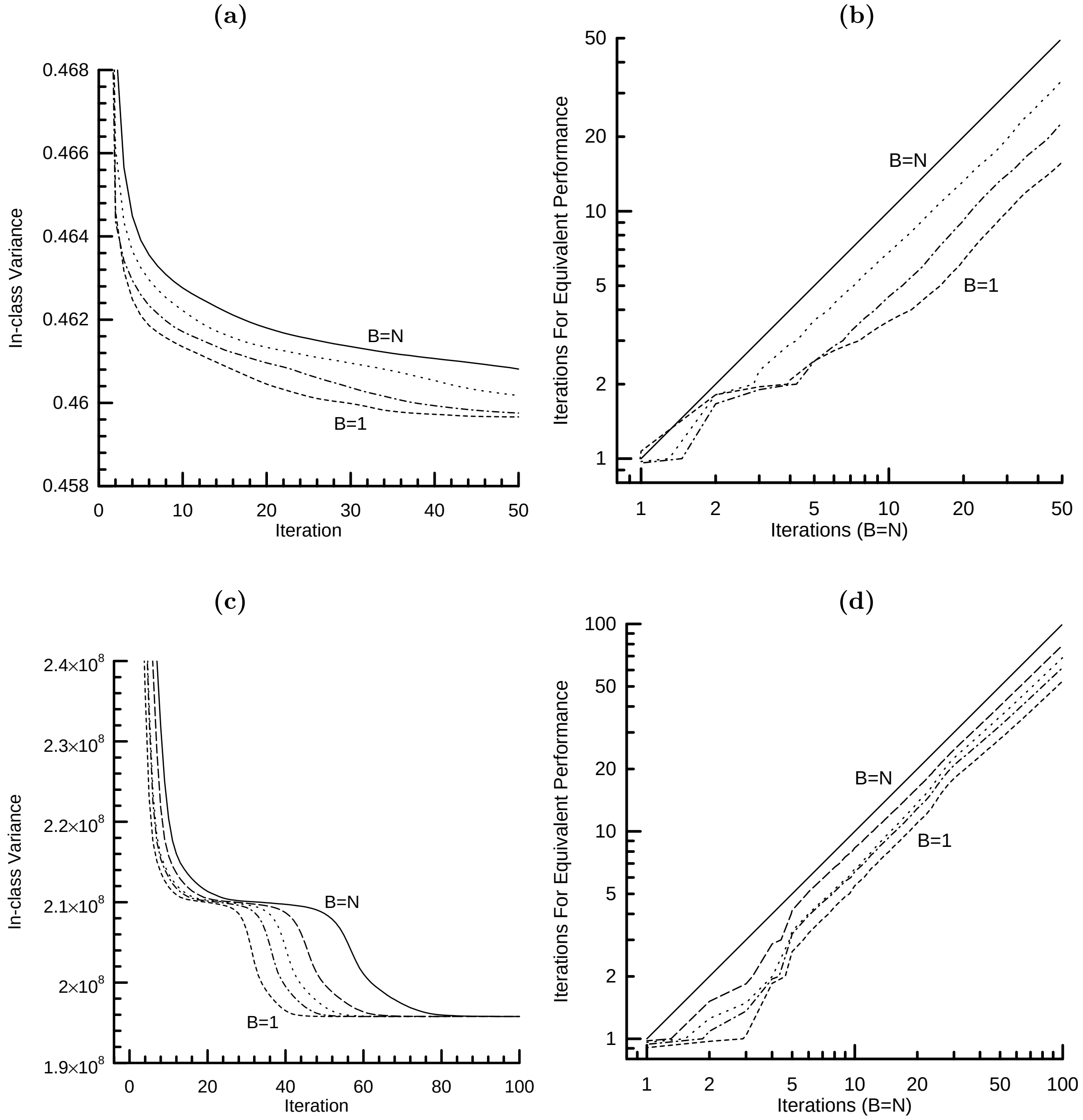

Figure 5. Performance of k-means clustering when updating centers after blocks of B data points. $B = N$ (solid line) corresponds to the traditional k-means algorithm, with centers updated after a full sweep through the data; $B = 1$ (dashed line) corresponds to updating centers "on the fly" (ie, every time a point is determined to change classes). **(a)** For uniformly distributed random data (see caption of Fig. 4(a)), updating on the fly is seen to produce faster convergence. The performance for intermediate values of B are shown with the dotted ($B = 5 \times 10^4$) and dashed-dotted ($B = 10^5$) lines. **(b)** Here, this faster convergence is explicitly shown – the vertical axis indicates the number of iterations required, for different values of B, to achieve the same performance as was achieved for the number of iterations shown on the horizontal axis using the standard $B = N$ approach. In this case, the on-the-fly approach produced the same quality clustering with roughly a quarter of the iterations required by the standard approach. **(c,d)** Same as in panels (a,b), but for the multispectral image data described in the caption to Fig. 4(b). Again, the on-the-fly approach was more efficient, but in this case, the gain was smaller than for the uniform random data. The intermediate values of B shown here are $B = 5 \times 10^4, 10^5, 2 \times 10^5$; for comparison $N \approx 3 \times 10^5$.

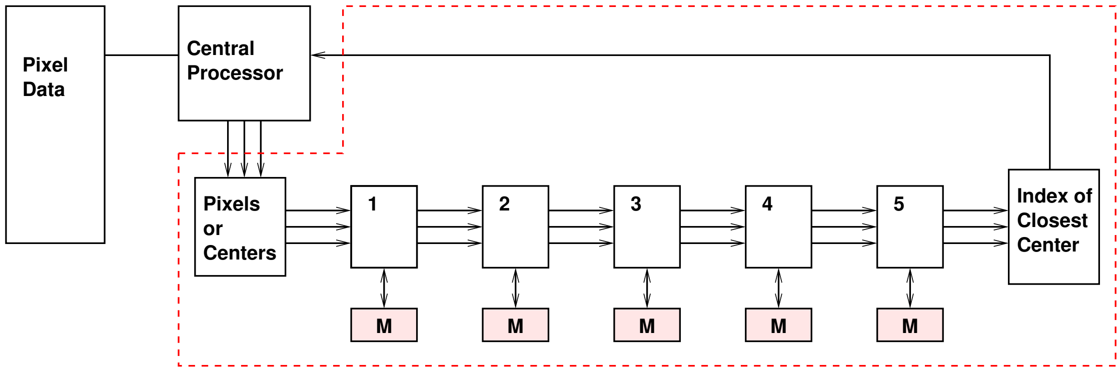

Figure 6. Data flow for k-means implementation on the hybrid processor. The central processor sends out a control bit to the hardware in order to indicate that the following data is center data or pixel data. When center data is indicated, the centers are loaded via the configurable logic to the embedded memory. When pixel data is selected the processor then sends out the block of B pixels. The index of the class which represents the minimum distance is returned to the central processor and it computes the new class centers, which it reloads into the embedded memory. Then it sends another block of B pixels, *etc.* until all the pixels have been evaluated.

as the rest of the chip. If the RISC processor were running on a faster clock (which is the ultimate goal for these hybrid systems), then the 11 cycles would go a lot faster and would not produce the problem they currently do.

A better way of sharing data between the processor and configurable logic is required. We have recently become aware of one such method, peculiar to this chip, that yields a latency of only 3 cycles to fetch data from memory for the configurable logic. This method observes the data and address bus as the software process is reading data and fetches data per a prescribed address map when a particular address location is observed. This method applied to the Kmeans algorithm could obtain an appreciable speed up of 3 to 4 times over our first mapping attempt.

2.3. Parameterizing the hardware implementation

The importance of parameterizing the hardware implementation is provided by an example given by Leeser *et al.*[61,59] There, an earlier implementation of k-means had hardcoded $K = 8$ clusters, $D = 10$ channels, and $b = 12$ bits per channel. A parameterized version was developed which treated these variables as parameters that could easily be changed. This obviously provides more flexibility for the user, but it is interesting to note that this flexibility did not come at the cost of decreased performance. For the same choice of K, D, and b, the parameterized design was actually smaller and virtually the same speed as the hardcoded design; furthermore, the place-and-route time was a factor of three smaller for the parameterized model.

2.4. Further approaches

One variant of the updating by blocks approach was suggested by Domingos and Hulten[62]; here a different block size is used at each iteration, with the size of the block based on estimates both of the misclassification error and of the centroid location error. An attempt is made to use large enough blocks to mimic (to within a prescribed ϵ) the behavior of the $B \to \infty$ convergence, but with $B \ll N$ points.

Kanugu *et al.*[63] describe a very sophisticated application of k-d trees to speed up the implementation of the k-means algorithm. The work achieves near order of magnitude speedup, at least for low-dimensional data, without introducing *any* approximations to the standard k-means algorithm. This is a "feature" but unfortunately it also makes it difficult to combine this approach with other speedups that do alter the basic algorithm. Translating the approach to a hardware implementation would be a challenge, to say the least.

The standard application of k-means to multispectral imagery treats the individual pixels as independent. A contiguity-enhanced approach has been described[64] which clusters the data in a way that produces segmentation with only slightly larger in-class variance, but substantially larger spatial contiguity for the classes. Ideally, we would like to impose this contiguity bias from the software, without altering the hardware design. This would give us the

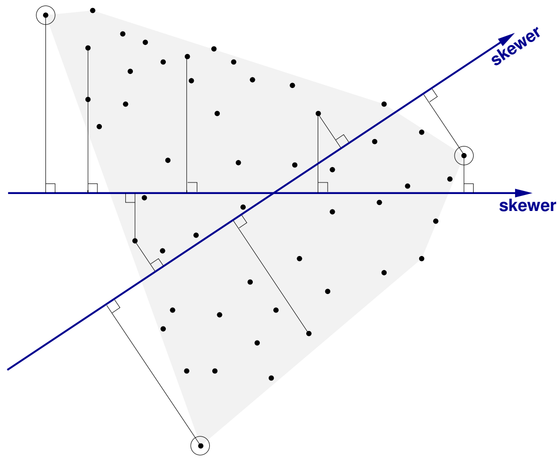

Figure 7. The PPI algorithm works by projecting points in the data set onto random skewers. For each skewer, two extreme points are identified, and their pixel purity index is incremented. In the figure above, the circled points are identified as candidate endmembers in the full space because their projection onto one or both of the skewers is extremal. The gray region indicates the convex hull of the points; only points on the convex hull are identified by the PPI algorithm.

most flexibility, but the contig-k-means algorithm is currently designed so that the bias is built into the distance function (which is computed in hardware). So adapting this problem to a co-design would also be a challenge.

When the desired number of clusters K is large, the standard k-means algorithm has a very slow converge rate, and many more iterations are required than would be for smaller values of K. One common approach is provided by the classic Linde-Buzo-Gray[65] approach; one begins with a small number of classes, and on subsequent iterations increases K. A systematic splitting algorithm was advocated by Fränti *et al.*[66] for faster clustering. Implemented in software, these algorithms achieve speedup in two ways: one is that fewer distance computations are required per data point at least for the early iterations, and the other is that fewer iterations are required. To take advantage of the first effect in hardware, one would have to re-design the configurable logic. But the fact that fewer iterations are required suggests that even a design that computes the sometimes-than-necessary K distance computations on every iteration will achieve a performance gain.

Once a clustering has been found, a split-and-merge algorithm[67] can be used to further improve the in-class variance by overcoming the problem of local minima. This will require some sophistication in the software to decide which clusters to split and which to merge, but with the distance computations farmed off to the highly parallel configurable logic, a good co-design could achieve the speedup benefits of both the hardware and the software, while at the same time achieving gains in the quality of the clustering.

3. ENDMEMBERS AND PIXEL PURITY INDEX

A number of algorithms have been proposed over the last decade for finding so-called endmembers in multispectral data.[68–78] The rationale behind endmembers is that a scene contains relatively few distinct materials and that much of the pixel-to-pixel variation in a multispectral image cube can be explained by the assumption that the pixels are mixtures of these distinct materials. The endmembers are the spectral signatures of these distinct materials, and the spectra of the rest of the pixels can be expressed as linear combinations (with positive weights, summing to unity) of these endmembers.

Endmembers are like principal components in that they provide a basis set for describing the rest of the data. But, because the endmembers are taken from the data themselves, and because the linear combinations are restricted to non-negative coefficients which sum to one, these endmembers are expected to provide a more "physical" basis than principal components.

The Pixel Purity Index (PPI) is one algorithm for identifying emdmembers from a multispectral dataset. The approach is to generate a large number n of random D-dimensional vectors (called "skewers" – see Fig. 7) and to

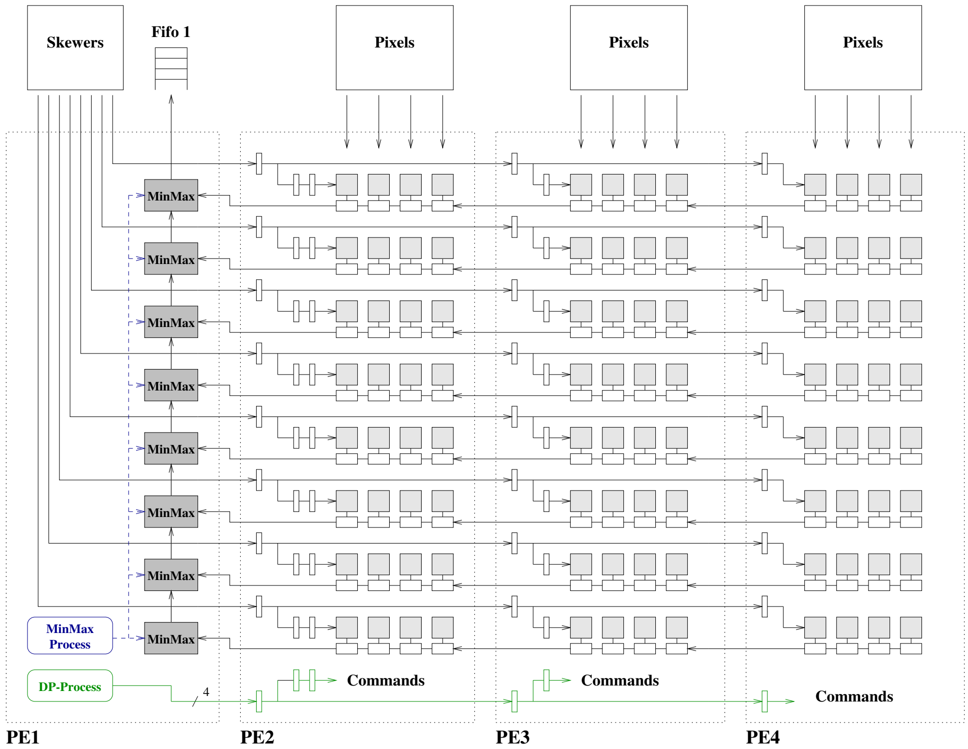

Figure 8. Architecture of an implementation of PPI on the Wildforce card.[79] The Wildforce contains four processing elements (PE1, PE2, PE3, PE4), each a distinct FPGA chip. This design uses three of the elements for computing dot products and the fourth for bookkeeping: routing skewer coefficients to the dot product components, computing extrema of the dot products and keeping track of the indices of pixels that produces those extreme, *etc.*

project the D-dimensional data onto them. A dot product is computed for every data point against every skewer, and the data points which correspond to extrema (or near extrema) in the direction of the skewer are identified, and placed on a list. As more skewers are generated, this list grows; the number of times a given pixel is placed on this list is also tallied. The pixels with the highest tallies are considered the most pure, and a pixel's count provides its "pixel purity index".

Lavenier *et al.*[79] describe an implementation of the PPI algorithm on a reconfigurable chip. See Fig. 8. A software approach for fast PPI was discussed by Theiler *et al.*[80] which expanded a small set of "direct dot products" into a larger set of "derived dot products." Although this approach was originally designed as a software speedup, and as such was a "competitor" to the hardware implementation, Fig. 9 shows how the software approach could be incorporated into the hardware design, combining the speedups from both approaches.

The pixel purity index was originally conceived[68] not as the full solution to the endmember problem, but as a a guide to be employed in an interactive way by a competent remote sensing scientist in conjunction with a good spectral library. A fully automated endmember identification algorithm might be desirable, but (with this approach, at least) is not something that one would want to design directly into hardware. However, the full endmember problem provides many opportunities for a co-design in which the software is responsible for directing the actual PPI in hardware, and for interpreting its results. The PPI algorithm, for instance, does not by itself identify a final list of endmembers; simply taking the $D + 1$ "most pure" pixels can lead to degenerate sets in which some of the endmembers are nearly identical. The choice of which pure pixels to choose as final endmembers is a postprocessing step that is better designed in software, where adjustments and modifications are easier to make. For instance, an otherwise expensive methodology with possibly complicated statistical properties (such as the archetypes of Cutler and Brieman[70]) could be more cheaply computed using the pixels with high purity index for initialization. The use of PPI to initialize the hypervolume maximizing algorithm (NFINDR) of Winter[78] has been described previously.[80] A closer interaction of the software and hardware would permit more complicated strategies for choosing the skewer directions based on endmembers that have been found so far – this might entail both exploration (in directions

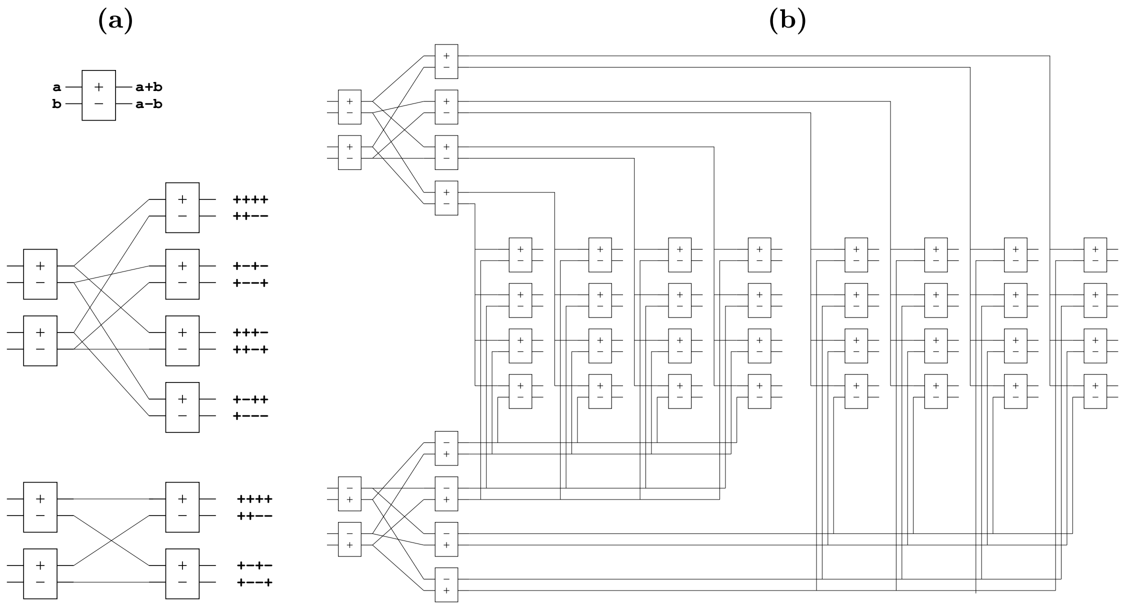

Figure 9. A schematic circuit for extending the PPI algorithm implemented in Fig. 8 to achieve eight-fold gain in the number of skewers without computing any more direct dot products. **(a)** The main building block is a module that takes two inputs and produces two outputs, corresponding to the sum and to the difference of the inputs. Combined into a circuit, six of these modules can take four inputs and produce eight outputs corresponding to all the sums and differences of the inputs. The positive and negative values of these eight sums correspond to the sixteen corners of a four-dimensional hypercube. Using only four modules, one can produce four outputs corresponding to the eight "alternate" corners of the four-dimensional hypercube. These alternates are all a Hamming distance of at least 2 from each other. **(b)** With 44 modules, one can leverage 8 inputs (direct dot products) into 64 outputs (derived dot products) corresponding to the alternate corners of an eight-dimensional hypercube. With this circuit element, one can transform the result of 8 direct dot products into 64 derived dot products. Applied to the Pixel Purity Index algorithm, this permits eight times as many skewers with only a fixed cost in terms of chip area. With respect to the design in Fig. 8, this circuit would fit between the dot products on PE 2-4 and the min-max computations on PE1. The eight min-max units on PE1 would have to be expanded to sixty four min-max units, but no extra dot products need be computed.

roughly perpendicular to existing endmembers) and refinement (with many nearly parallel skewers in the direction of existing endmembers).

4. CONCLUSION

If an algorithm is a procedure for getting something done, then much of the art in algorithm design depends on knowing about that "something." In contrast to the detailed specifications that go into the *implementation* of an algorithm, the ultimate *aim* of an algorithm is often just a little bit fuzzy. Some classical algorithms (eg, sorting an array, computing a Fourier Transform, or finding the convex hull) are indeed well-specified, but the algorithms that one needs in the real world, particularly in the real world of remote sensing, do not always have such mathematically well-defined goals.

While mathematical expressions of those goals are an important starting point, one must sometimes step back and remember what the real goals are. Clustering puts spectrally similar pixels into the same class, and has the effect of reducing the within-class variance. But the mathematical goal of finding *the* partition with the global minimum of within-class variance is a combinatorially difficult problem. On the other hand, by using an iterative algorithm (k-means) with a simplified distance measure (Manhattan) on precision-truncated data, one can very rapidly compute an approximation to the optimal solution. This approximation may be suboptimal from the mathematical point of view, but it does achieve the real goal – that spectrally similar pixels be classified together. More important than squeezing the last percentage point of optimality from the mathematical constraints is to reconsider the physical phenomena (such as correlations between spectral channels or the tendency for contiguous pixels to be in the same class) that drive the mathematical formulation.

By the same token, while the concepts of convex geometry provide useful insight into the problem of detecting endmembers in imagery, we have to keep in mind that the endmember concept itself is an approximation: real scenes are not really composed of a small number of distinct materials, nor are pixels exact linear combinations of of these materials. The convex hull is a useful geometrical visualization of what we desire to estimate from our data, but we do not literally want to find the convex hull of a million points in a hundred dimensional space.

Remote sensing is a complicated science; and algorithms for remote sensing retrievals will necessarily be complicated as well, if they are to achieve a useful level of physical fidelity. Complicated algorithms do not fit well in hardware, and are a waste of valuable real estate that could be used for further fine-grained parallelism. We do believe that reconfigurable hardware has a place in remote sensing computation, particularly for real-time processing. But to produce fast implementations of complicated and flexible algorithms will require the simultaneous design of some components that run well in massively parallel mode on reconfigurable hardware, some components that run well on general purpose serial processors, and appropriate interconnections that keep both talking to each other without overloading the limited bandwidth between them.

ACKNOWLEDGMENTS

We are pleased to acknowledge many stimulating conversations with Miriam Leeser and with Dominique Lavenier on this topic. This work was supported by the United States Department of Energy, through the Deployable Adaptive Processing Systems (DAPS) project at Los Alamos National Laboratory.

REFERENCES

1. R. Sedgewick and C. J. Van Wyk, *Algorithms in C++*, Addison-Wesley, Reading, Massachusetts, 1998.
2. Xilinx, Inc. `http://www.xilinx.com`.
3. Altera Corporation. `http://www.altera.com`.
4. Annapolis Micro Systems, Inc. `http://www.annapmicro.com`.
5. J. Villasenor and W. H. Mangione-Smith, "Configurable computing," *Scientific American* , pp. 66–71, June 1997.
6. P. Banerjee, N. Shenoy, A. Choudhary, S. Hauck, C. Bachmann, M. Haldar, P. Joisha, A. Jones, A. Kanhare, A. Nayak, S. Periyacheri, M. Walkden, and D. Zaretsky, "A MATLAB compiler for distributed, hetergeneous, reconfigurable computing systems," in *IEEE Symposium on FPGAs for Custom Computing Machines*, K. L. Pocek and J. Arnold, eds., pp. 39–48, IEEE Computer Society Press, 2000.

7. M. B. Gokhale, J. M. Stone, J. Arnold, and M. Kalinowski, “Stream-oriented FPGA computing in the Streams-C high level language,” in *IEEE Symposium on FPGAs for Custom Computing Machines*, K. L. Pocek and J. Arnold, eds., pp. 49–58, IEEE Computer Society Press, 2000.
8. B. Draper, W. Najjar, W. Bohm, J. Hammes, B. Rinker, C. Ross, M. Chawathe, and J. Bins, “Compiling and optimizing image processing algorithms for FPGAs,” in *International Workshop on Computer Architectures for Machine Perception (CAMP)*, 2000.
9. M. P. Caffrey, A. Begtrup, J. Layne, T. Nelson, S. Robinson, A. Salazar, J. J. Szymanski, and J. Theiler, “High performance signal and image processing for remote sensing using reconfigurable computers,” *Proc. SPIE* **3807**, pp. 142–149, 1999.
10. P. V. Villeneuve, H. A. Fry, J. Theiler, and B. W. Smith, “Improved matched filter detection techniques,” *Proc. SPIE* **3753**, pp. 278–285, 1999.
11. A. Nelson and K. McCabe, “Flexible architecture for hyperspectral image processing on reconfigurable computers,” Tech. Rep. LA-UR-01-2900, Los Alamos National Laboratory, 2001.
12. R. Porter, M. Gokhale, N. Harvey, S. Perkins, and C. Young, “Evolving network architectures with custom computers for multi-spectral feature identification,” in *3rd NASA/DoD Workshop on Evolvable Hardware*, July 2001.
13. R. B. Porter, M. Gokhale, N. R. Harvey, S. J. Perkins, and C. Young, “Evolving a spatio-spectral network on reconfigurable computing for multispectral feature identification,” *Proc. SPIE* **4480**, 2001. (in this volume).
14. See `http://www.daps.lanl.gov/genie` for a description of the GENetic Imagery Exploitation (GENIE) system, as well as a list of references.
15. J. R. Hauser and J. Wawrzynek, “GARP: A MIPS processor with a reconfigurable coprocessor,” in *IEEE Symposium on FPGAs for Custom Computing Machines*, J. Arnold and K. L. Pocek, eds., pp. 12–21, IEEE Computer Society Press, 1997.
16. C. Rupp, M. Landguth, T. Garverick, E. Gomersall, H. Holt, J. Arnold, and M. Gokhale, “The Napa Adaptive Processing Architecture,” in *IEEE Symposium on FPGAs for Custom Computing Machines*, K. L. Pocek and J. Arnold, eds., pp. 28–37, IEEE Computer Society Press, 1998.
17. M. B. Gokhale and J. M. Stone, “Co-synthesis to a hybrid RISC/FPGA architecture,” *Journal of VLSI Signal Processing Systems* **24**, pp. 165–180, 2000.
18. Altera Corporation. `http://www.altera.com/products/devices/excalibur/exc-index.html`.
19. Xilinx, Inc. `http://www.xilinx.com/prs_rls/ibmpartner.htm`.
20. Quicksilver Technology, Inc. `http://www.quicksilvertech.com`.
21. Triscend Corporation. `http://www.triscend.com`.
22. Adaptive Silicon, Inc. `http://www.adaptivesilicon.com`.
23. Integrated Circuit Technology Corporation. `http://www.ictpld.com`.
24. Infinite Technology Corporation. `http://www.itc-usa.com`.
25. LightSpeed Semiconductor. `http://www.lightspeed.com`.
26. MorphICs Technology, Inc. `http://www.morphics.com`.
27. Systolix. `http://www.systolix.co.uk/nf/index.htm`.
28. M. Gokhale, J. Frigo, K. McCabe, J. Theiler, and D. Lavenier, “Early experience with a hybrid processor: k-means clustering,” in *ERSA '01: First International Conference on Engineering of Reconfigurable Systems and Algorithms (Las Vegas, NV, 25-28 June 2001)*, 2000.
29. E. Forgy, “Cluster analysis of multivariate data: efficiency versus interpretability of classifications,” *Biometrics* **21**, p. 768, 1965. (Abstract).
30. G. H. Bull and D. J. Hall, “ISODATA – a novel method of data analysis and pattern classification,” tech. rep., Stanford Research Institute, Menlo Park, CA, 1965.
31. J. MacQueen, “On convergence of k-means and partitions with minimum average variance,” *Ann. Math. Statist.* **36**, p. 1084, 1965. (Abstract).
32. J. MacQueen, “Some methods of classification and analysis of multivariate observations,” in *Proceedings of Fifth Berkeley Symposium on Mathematical Statistics and Probability*, L. M. L. Cam and J. Neyman, eds., pp. 281–297, University of California Press, (Berkeley), 1967.

33. S. P. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Information Theory* **IT-28**, pp. 129–137, 1982. This article represents the first published appearance of Lloyd's 1957 manuscript, written in the Mathematical Research Department at Bell Laboratories.
34. L. M. Ni and A. K. Jain, "A VLSI systolic architecture for pattern clustering," *IEEE Trans. on Pattern Anal. and Mach. Intel. PAMI* **7**, pp. 80–89, 1985.
35. A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. Roy. Stat. Soc. B* **39**, pp. 1–38, 1977.
36. J. A. Hartigan, *Clustering Algorithms*, John Wiley & Sons, New York, 1975.
37. H. Späth, *Cluster Analysis Algorithms for Data Reduction and Classification of Objects*, John Wiley & Sons, New York, 1975.
38. R. M. Gray, "Vector quantization," *IEEE ASSP Mag.* **1**, pp. 4–29, 1984.
39. H. Späth, *Cluster Dissection and Analysis: Theory, FORTRAN programs, Examples*, Ellis Horwood Limited, Chichester, UK, 1985.
40. A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, NJ, 1988.
41. A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Computing Surveys* **31**, pp. 264–323, 1999.
42. P. M. Kelly and J. M. White, "Preprocessing remotely-sensed data for efficient analysis and classification," *Proc. SPIE* **1963**, pp. 24–30, 1993.
43. R. A. Schowengerdt, *Techniques for Image Processing and Classification in Remote Sensing*, Academic Press, Orlando, 1983.
44. D. M. Titterington, "Updating a diagnostic system using unconfirmed cases," *Appl. Statist.* **25**, pp. 238–247, 1976.
45. T. J. O'Neil, "Normal discrimination with unclassified observations," *J. Amer. Statist. Assoc.* **73**, pp. 821–826, 1978.
46. G. J. McLachlan and S. Ganesalingam, "Updating the discriminant function on the basis of unclassified data," *Communications in Statistics – Simulation and Computation* **11**, pp. 753–767, 1982.
47. V. Castelli and T. M. Cover, "On the exponential value of labeled samples," *Pattern Recognition Lett.* **16**, pp. 105–111, 1995.
48. 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.
49. A. Blum and S. Chawla, "Learning from labeled and unlabeled data using graph minicuts," in *Proceedings of the 18th International Conference on Machine Learning*, C. E. Brodley and A. P. Danyluk, eds., pp. 19–26, Morgan Kaufmann, (San Francisco), 2001.
50. D. G. Sheppard, A. Bilgin, M. S. Nadar, B. R. Hunt, and M. W. Marcellin, "Vector quantizer for image restoration," *IEEE Trans. Image Processing* **7**, pp. 119–124, 1998.
51. C. Funk, J. Theiler, D. A. Roberts, and C. C. Borel, "Clustering to improve matched-filter detection of weak gas plumes in hyperspectral imagery," *IEEE Trans. Geosci. Remote Sensing* , 2001. In press.
52. A. D. Stocker and A. P. Schaum, "Application of stochastic mixing models to hyperspectral detection problems," *Proc. SPIE* **3071**, pp. 47–60, 1997.
53. M. E. Leeser, J. Theiler, M. Estlick, N. V. Kitaryeva, and J. J. Szymanski, "Effect of data truncation in an implementation of pixel clustering on a custom computing machine," *Proc. SPIE* **4212**, pp. 80–89, 2000.
54. NASA, 1999. `http://makalu.jpl.nasa.gov/avaris.html`.
55. P. G. Weber, B. C. Brock, A. J. Garrett, B. W. Smith, C. C. Borel, W. B. Clodius, S. C. Bender, R. R. Kay, and M. L. Decker, "MTI Mission Overview," *Proc. SPIE* **3753**, pp. 340–346, 1999.
56. J. Theiler, M. Leeser, M. Estlick, and J. J. Szymanski, "Design issues for hardware implementation of an algorithm for segmenting hyperspectral imagery," *Proc. SPIE* **4132**, 2000.
57. M. Leeser, J. Theiler, M. Estlick, and J. J. Szymanski, "Design tradeoffs in a hardware implementation of the k-means clustering algorithm.," in *Proc. SAM 2000: First IEEE Sensor Array and Multichannel Signal Processing Workshop, 16-17 March 2000, Cambridge, MA.*, pp. 520–524, 2000.
58. M. Estlick, M. Leeser, J. Theiler, and J. J. Szymanski, "Algorithmic transforms in the implementation of k-means clustering on reconfigurable hardware," in *FPGA 2001: Ninth International Symposium on Field Programmable Gate Arrays*, pp. 103–110, Association for Computing Machinery, 2001.

59. M. Leeser, P. Belanovic, M. Estlick, M. Gokhale, J. J. Szymanski, and J. Theiler, "Applying reconfigurable hardware to the analysis of multispectral and hyperspectral imagery," *Proc. SPIE* **4480**, 2001.
60. V. Faber, "Clustering and the continuous k-means algorithm," *Los Alamos Science* **22**, pp. 138–144, 1994.
61. M. Leeser, P. Belanovic, M. Estlick, M. Gokhale, J. J. Szymanski, and J. Theiler, "Parameterized k-means clustering for rapid hardware development to accelerate analysis of satellite data," in *High Performance Embedded Computing (HPEC) Workshop, MIT Lincoln Laboratory*, 2001.
62. P. Domingos and G. Hulten, "A general method for scaling up machine learning algorithms and its application to clustering," in *Proceedings of the 18th International Conference on Machine Learning*, C. E. Brodley and A. P. Danyluk, eds., pp. 106–113, Morgan Kaufmann, (San Francisco), 2001.
63. T. Kanungu, D. M. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu, "The analysis of a simple k-means clustering algorithm," *Proc. 16th ACM Symp. on Computational Geometry* , pp. 101–109, 2000. An updated version, entitled "An Efficient k-Means Clustering Algorithm: Analysis and Implementation" is available at `http:///www.cs.umd.edu/∼mount/pubs.html`.
64. J. Theiler and G. Gisler, "A contiguity-enhanced k-means clustering algorithm for unsupervised multispectral image segmentation," *Proc. SPIE* **3159**, pp. 108–118, 1997.
65. Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Communications* **COM-28**, pp. 84–95, 1980.
66. P. Fränti, T. Kaukoranta, and O. Nevalainen, "On the splitting method for VQ codebook generation," *Opt. Eng.* **36**, pp. 3043–3051, 1997.
67. T. Kaukoranta, P. Fränti, and O. Nevalainen, "Iterative split-and-merge algorithm for vector quantization codebook generation," *Opt. Eng.* **37**, pp. 2726–2732, 1998.
68. J. W. Boardman, "Automating spectral unmixing of AVIRIS data using convex geometry concepts," in *Summaries of the Fourth Annual JPL Airborne Geoscience Workshop*, R. O. Green, ed., pp. 11–14, 1994.
69. J. W. Boardman, "Geometric mixture analysis of imaging spectrometry data," *Proc. IGARSS (IEEE International Geoscience and Remote Sensing Symposium)* , pp. 2369–2371, 1994.
70. A. Cutler and L. Breiman, "Archetypal analysis," *Technometrics* **36**, pp. 338–347, 1994.
71. J. W. Boardman, F. A. Kruse, and R. O. Green, "Mapping target signatures via partial unmixing of AVIRIS data," in *Summaries of Fifth Annual JPL Airborne Earth Science Workshop*, R. O. Green, ed., pp. 23–26, 1995.
72. D. A. Roberts, M. Gardner, R. Church, S. Ustin, G. Scheer, and R. O. Green, "Mapping chaparral in the Santa Monica mountains using multiple spectral mixture models," in *Summaries of 6th JPL Airborne Earth Science Workshop*, R. O. Green, ed., pp. 197–201, 1996.
73. S. Tompkins, J. F. Mustard, C. M. Pieters, and D. W. Forsyth, "Optimization of endmembers for spectral mixture analysis," *Remote Sensing of the Environment* **59**, pp. 472–489, 1997.
74. G. S. Okin, W. J. Okin, D. A. Roberts, and B. Murray, "Multiple endmember spectral mixture analysis: Application to an arid/semi-arid landscape," in *Summaries of the Seventh JPL Airborne Earth Science Workshop*, R. O. Green, ed., pp. 291–299, 1998.
75. C. A. Bateson, G. P. Asner, and C. A. Wessman, "Incorporating endmember variability into spectral mixture analysis through endmember bundles," in *Summaries of the Seventh JPL Airborne Earth Science Workshop*, R. O. Green, ed., pp. 43–52, 1998.
76. J. Bowles, M. Daniel, J. Grossman, J. Antoniades, M. Baumback, and P. Palmadesso, "Comparison of output from ORASIS and pixel purity calculations," *Proc. SPIE* **3438**, pp. 148–156, 1998.
77. A. Mooijaart, P. G. M. van der Heijden, and L. A. van der Ark, "A least squares algorithm for a mixture model for compositional data," *Computational Statistics and Data Analysis* **30**, pp. 359–379, 1999.
78. M. E. Winter, "N-FINDR: an algorithm for fast autonomous spectral end-member determination in hyperspectral data," *Proc. SPIE* **3753**, pp. 266–277, 1999.
79. D. D. Lavenier, J. Theiler, J. J. Szymanski, M. Gokhale, and J. R. Frigo, "FPGA implementation of the pixel purity index algorithm," *Proc. SPIE* **4212**, 2000.
80. J. Theiler, D. D. Lavenier, N. R. Harvey, S. J. Perkins, and J. J. Szymanski, "Using blocks of skewers for faster computation of pixel purity," *Proc. SPIE* **4132**, pp. 61–71, 2000.