Using blocks of skewers for faster computation of Pixel Purity Index

James Theiler, Dominique D. Lavenier, Neal R. Harvey,
Simon J. Perkins, and John J. Szymanski

Space and Remote Sensing Sciences Group
Los Alamos National Laboratory
Los Alamos, New Mexico 87545

ABSTRACT

The "pixel purity index" (PPI) algorithm proposed by Boardman, *et al.*[1] identifies potential endmember pixels in multispectral imagery. The algorithm generates a large number of "skewers" (unit vectors in random directions), and then computes the dot product of each skewer with each pixel. The PPI is incremented for those pixels associated with the extreme values of the dot products. A small number of pixels (a subset of those with the largest PPI values) are selected as "pure" and the rest of the pixels in the image are expressed as linear mixtures of these pure endmembers. This provides a convenient and physically-motivated decomposition of the image in terms of a relatively few components.

We report on a variant of the PPI algorithm in which blocks of B skewers are considered at a time. From the computation of B dot products, one can produce a much larger set of "derived" dot products that are associated with skewers that are linear combinations of the original B skewers. Since the derived dot products involve only scalar operations, instead of full vector dot products, they can be very cheaply computed.

We will also discuss a hardware implementation on a field programmable gate array (FPGA) processor both of the original PPI algorithm and of the block-skewer approach. We will furthermore discuss the use of fast PPI as a front-end to more sophisticated algorithms for selecting the actual endmembers.

Keywords: Hyperspectral, Multispectral, Endmembers, Remote Sensing, Pixel Purity, Field Programmable Gate Array (FPGA)

1. INTRODUCTION

A number of algorithms have been proposed over the last decade for finding so-called endmembers in multispectral data.[1–11] These algorithms are all based on the notion 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 "pure" materials. The endmembers are the spectral signatures of these pure materials.

The "spectral unmixing" problem actually involves two steps. The first step is to find the endmember signatures (either from a library of known spectra or directly from the image), and the second is to express the individual pixels as (usually linear) combinations of these endmembers. The coefficients associated with each of the endmembers are identified with the "abundances" of the endmember materials in the given pixel; the coefficients are generally assumed to be proportional to the areal extent of each of the endmember materials in the pixel.

The emphasis of our work is on the computation involved in the first step, the identification of endmembers, directly from an image.

Like principal components, endmembers provide a basis set in terms of which the rest of the data can be described. But unlike principal components, the endmembers are expected to provide a more "physical" description of the data. There are two reasons for this expectation: One is that the endmembers are taken from the data themselves, and the other is that the linear combinations are restricted to combinations in which the coefficients are non-negative and sum to one. The mixed pixels can be literally interpreted as having X-percent of material A, and Y-percent of material B.

Emails: {jt,lavenier,harve,s.perkins,szymanski}@lanl.gov.

2. ORIGINAL PPI ALGORITHM

The importance of convex geometry as a paradigm for understanding the endmember problem was emphasized by Boardman *et al.*[1–3] early in this decade. If a multispectral image has D spectral channels, then each pixel can be identified with a point in a D-dimensional space.

Boardman's "pixel purity index" (PPI) algorithm[1] represents a specific attempt to exploit this paradigm, by locating data points which are on the vertices of the convex hull of this D-dimensional scatterplot. The problem of identifying the convex hull from a discrete set of data is a classic one in computational geometry, but the PPI algorithm provides a number of advantages over these classical algorithms. First, as we will describe below, it is relatively simple to implement, and it parallelizes quite naturally. Furthermore, it produces a relative measure for each vertex which distinguishes those near "corners" of the data with a higher-valued index: this is important because the corners are where the endmembers are expected to be. Although it is an approximate algorithm (it cannot be guaranteed to identify every vertex except in an asymptotic limit), it produces approximate answers right away, and the approximation gets better as the algorithm progresses. Finally, the penalty for working in a high-dimensional space increases roughly linearly (rather than exponentially) with the nominal dimension D of the space.

The algorithm proceeds by generating a large number N of random D-dimensional "skewers" through the D-dimensional data. For each skewer, every data point is projected onto the skewer, and the position along the skewer is is noted. 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".

The PPI algorithm, by itself, does not identify a final list of endmembers. Taking the $D+1$ "most pure" pixels, for instance, often leads to degenerate sets in which some of the endmembers are nearly identical. The pixel purity index was conceived not as a solution, but as a a guide; the author[2] proposed comparing the pure pixels with target spectra from a library, and successively projecting the data to lower dimensional spaces as endmembers were identified. Nonetheless, as Fig. 2 shows, the PPI algorithm can identify a small set of candidate endmembers from a large image. To illustrate the use of PPI and its variants, we use a 614×512-pixel 224-channel hyperspectral AVIRIS image[12]; we also use an image which is derived from the AVIRIS image, but only has ten channels, corresponding to bands available on the MTI satellite.[13]

Since PPI identifies more than just the $D+1$ points that make up the optimal simplex, it can potentially provide useful information to other algorithms (such as Archetypal Analysis[4] or N-FINDR[11]) to accelerate their convergence. This puts the PPI algorithm in the role of image pre-processor, and argues further in favor of its choice for hardware acceleration.

3. BLOCKS OF SKEWERS

The original PPI algorithm generates a large number of "skewers" (unit vectors in random directions), and then computes the dot product of each skewer with each data point. This can be computationally expensive, especially in high dimensions, and requires that a large number of dot products be evaluated.

Consider a block of B skewers, $\mathbf{k}_b \in \mathcal{R}^D$ with $b = 1, \ldots B$, and consider the B scalar dot products $d_b \in \mathcal{R}$ obtained from each of these skewers applied to one of the data points. The central observation in the blocks-of-skewers method is that for every skewer which is a linear combination of these B skewers, the "derived" dot product of that skewer with the data is a linear combination of the original dot products d_b. The reason this is useful is that derived dot products can be much cheaper to compute than original dot products. One price paid for this convenience is that the virtual skewers that produce the derived dot products are in the same B-dimensional subspace as the original B skewers. However, with appropriate choice of B and with a reasonable strategy for choosing the linear combinations, a considerable computational advantage can be obtained.

Given a set of B skewers, $\mathbf{k}_i \in \mathcal{R}^D$, for $i = 1, \ldots, B$; consider the (potentially much larger) set of virtual skewers given by the linear combinations $\mathbf{k}' = a_1\mathbf{k}_1 + \ldots + a_B\mathbf{k}_B$. There is in principle an infinite choice of coefficients a_i, $i = 1, \ldots, B$, but we will consider various restrictions below. But even for arbitrary $a_i \in \mathcal{R}$, already there is a computational advantage that can be seen.

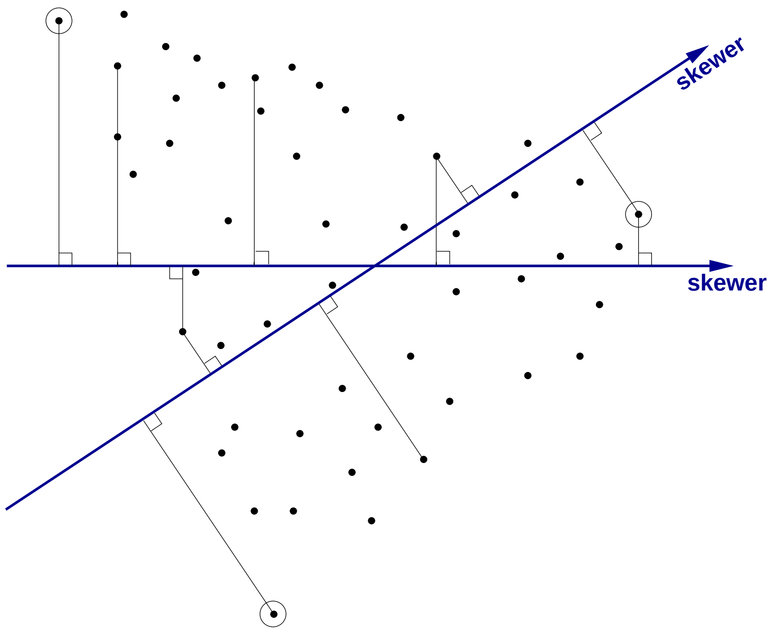

Figure 1. The PPI algorithm works by projecting points in the data set onto random skewers. For each skewer, two extreme points are identified, and their pixel purity index is incremented. In the figure above, the circled points are identified as candidate endmembers in the full space because their projection onto one or both of the skewers is extremal.

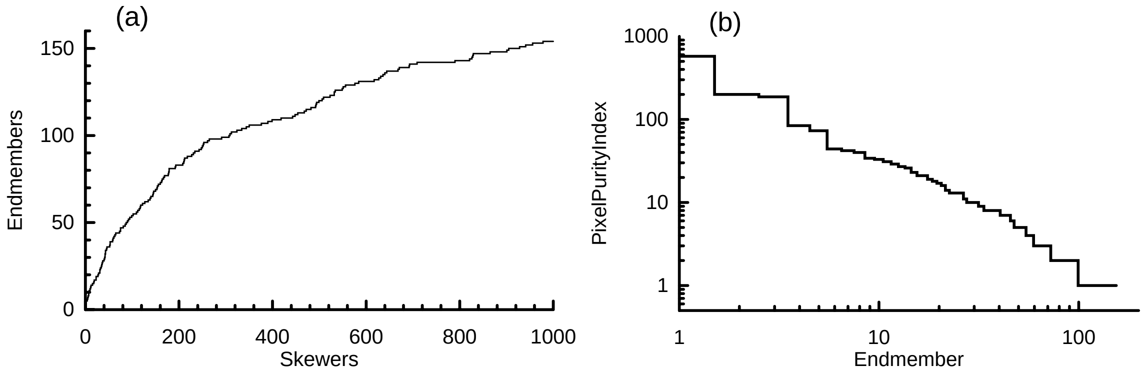

Figure 2. **(a)** Plot of the cumulative number of endmembers found as a function of the number of skewers in a run of PPI on a ten-channel multispectral data set. After the run of $N = 1000$ skewers, a total of 154 candidate endmembers were produced. **(b)** Sorting the candidate endmembers according to their pixel purity index shows the range of pixel purity indices exhibited by the candidate endmembers: a few have very large values, some have intermediate values, and fully a third of them have an index of unity. Those points with higher indices are presumed to be closer to the "corners" of the data in the ten-dimensional space.

(a)

$$
\begin{array}{ccccc}
p_1 & & & & \\
& & p_2 & & \\
p_1 & + & p_2 & & \\
p_1 & - & p_2 & & \\
& & & & p_3 \\
p_1 & & & + & p_3 \\
& & p_2 & + & p_3 \\
p_1 & + & p_2 & + & p_3 \\
p_1 & - & p_2 & + & p_3 \\
p_1 & & & - & p_3 \\
& & p_2 & - & p_3 \\
p_1 & + & p_2 & - & p_3 \\
p_1 & - & p_2 & - & p_3 \\
& & \vdots & &
\end{array}
$$

(b)

$$
\begin{array}{ccccccc}
p_1 & + & p_2 & + & p_3 & + & \cdots \\
p_1 & - & p_2 & + & p_3 & + & \cdots \\
p_1 & - & p_2 & - & p_3 & + & \cdots \\
p_1 & + & p_2 & - & p_3 & + & \cdots \\
p_1 & + & p_2 & - & p_3 & - & \cdots \\
p_1 & - & p_2 & - & p_3 & - & \cdots \\
p_1 & - & p_2 & + & p_3 & - & \cdots \\
p_1 & + & p_2 & + & p_3 & - & \cdots \\
& & & \vdots & & &
\end{array}
$$

Figure 3. If $p_1, p_2, p_3, \ldots$ are the original dot products, computed from a block of skewers applied to a single data point, then the derived dot products are linear combinations of these: $a_1 p_2 + a_2 p_2 + a_3 p_3 + \cdots$. Appropriately ordering the derived dot product calculations can lead to considerably reduced computation at each step. **(a)** For the simple discrete case in which $a_i \in \{-1, 0, 1\}$, each of the derived dot products associated with the $O(3^B)$ virtual skewers are obtained by a single addition or subtraction to a term that has been computed previously. **(b)** For the corners of the hypercube case, with $a_i \in \{-1, 1\}$, an arrangement like a graycode leads to a set of computations in which each term is equal to the previous term plus or minus twice one of the original dot products.

For each of the D-dimensional data points $\mathbf{x_n}$, the B dot products $\mathbf{k}_i \cdot \mathbf{x_n}$, for $i = 1, \ldots, B$. each require D multiply-accumulate operations. But the derived dot product

$$\mathbf{k}' \cdot \mathbf{x_n} = a_1 \mathbf{k}_1 \cdot \mathbf{x_n} + \ldots + a_B \mathbf{k}_B \cdot \mathbf{x_n} \tag{1}$$

can be expressed as a linear combination of these B dot products, and can be computed with only B multiply-accumulates. As long as $B < D$, these derived dot products are cheaper to compute than the original dot products; on the other hand, the derived dot products are with vectors that are in the B-dimensional subspace spanned by $\mathbf{k}_i$ for $i = 1, \ldots, B$.

3.1. Discrete Linear Combinations

If, rather than permit arbitrary coefficients, we restrict them to be small discrete integers, *i.e.,* $a_i \in \{-n, -(n-1), \ldots, -1, 0, 1, \ldots, n-1, n\}$, then multiplication by a_i can be implemented as a small number of additions. In this case, then there are $(2n+1)^B$ of these virtual skewers, though the useful count is $((2n+1)^B - 1)/2$, neglecting the $\mathbf{k}' = 0$ skewer and recognizing that the overall sign of the skewer doesn't matter (since we consider both maximum and minimum values of the dot products).

Here, even $n = 1$ provides an $O(3^B)$ gain, though not all of this exponential gain can be realized in practice. The calculation of each derived dot product nominally requires $O(B)$ additions, though by computing the derived dot products in a well-chosen order, this number can be reduced to $O(1)$; see Fig. 3(a). The price per dot product can be greatly reduced, but the dot products must still be applied to every data point. And for each skewer, virtual or otherwise, one must keep track of maximum and minimum dot product, as the algorithm is looped over the entire data set.

The speedup that is available from these virtual skewers is considerable. For a high dimensional space, it is the multiply-accumulates in the original dot products that will dominate the computation, and since we get $(3^B - 1)/2$

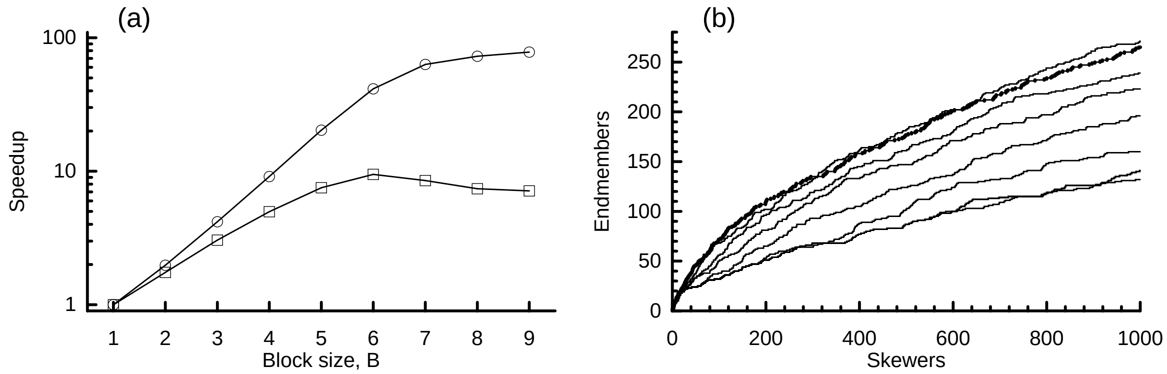

Figure 4. **(a)** Speedup available from the block-skewer method increases rapidly with block size B, but saturates at a speedup that scales with the dimension D of the data. The circles represent the speedup attained when the block-skewer method was applied to find endmembers of a 224-channel AVIRIS image; the squares are the same calculation, but applied to a 10-channel simulated MTI image. In both cases, discrete linear combinations with $a_i \in \{-1, 0, 1\}$ were used. **(b)** A price is paid for this speedup. Shown here are the cumulative endmembers found as a function of number of skewers. The darker line corresponds to the standard (slow) $B = 1$ algorithm, and the performance generally decreases with increasing B.

virtual skewers out of our B original skewers, the theoretical speedup is

$$S_o = \frac{3^B - 1}{2B}. \tag{2}$$

However, if we account for the time required for an addition or subtraction ($t_{\text{add/sub}}$), which is needed for every virtual skewer, the time required to maintain a record of the minimum and maximum dot product so far ($t_{\text{min/max}}$), then we obtain a formula for speedup

$$S = \frac{S_o}{1 + S_o \tau / D}, \tag{3}$$

where D is the number of spectral channels, and τ is the ratio

$$\tau = \frac{t_{\text{add/sub}} + t_{\text{min/max}}}{t_{\text{mult/acc}}}. \tag{4}$$

For large D, this looks like the idealized speedup S_o until it saturates at $S_{\text{top}} = 1/(D\tau)$.

3.2. Corners of the hypercube

If we further restrict consideration to virtual skewers $\mathbf{k}' = \sum_{i=1}^{B} a_i \mathbf{k}_i$, with $a_i \in \{-1, 1\}$, then there are 2^{B-1} derived dot products for every B original dot products. This is not as large as the $O(3^B)$ gain seen in the previous section, but the gain is still exponential.

The advantage comes in comparing the angular separation of nearby skewers. If $\mathbf{k}_1$ and $\mathbf{k}_2$ are two skewers, then

$$\cos\theta = \frac{\mathbf{k}_1' \cdot \mathbf{k}_2'}{|k_1'|\ |k_2'|} \tag{5}$$

defines the angle between them. The closest pairs of skewers will be those whose a_i values differ only at a single i. For those pairs, assuming the original B skewers are orthogonal, the angle between them will be given by

$$\cos\theta = 1 - \frac{2}{B} \tag{6}$$

By contrast the closest pair of skewers when a_i is permitted to include 0 as well as $\{-1, 1\}$ is given by

$$\cos\theta = \frac{B-1}{\sqrt{(B-1)B}} \approx 1 - \frac{1}{2B}. \tag{7}$$

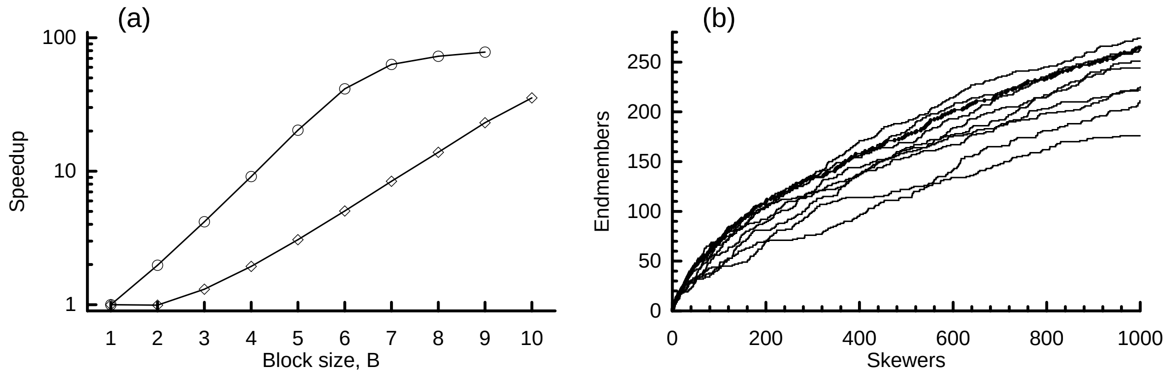

Figure 5. **(a)** Speedup for two different block-skewer methods; the first (circles) is the discrete linear combinations with $a_i \in \{-1, 0, 1\}$ were used, which is the same as shown in Fig. 4(a). The second is the alternate corners method, with $a_i \in \{-1, 1\}$, and only every other corner used. The speedup of the alternate corners method does not increases as rapidly with block size B, but it also does not saturate for as small a value of B. **(b)** The real advantage of the alternate corners method, compared to all discrete linear combinations with $a_i \in \{-1, 0, 1\}$, is seen in the plot of cumulative endmembers found as a function of number of skewers. Comparing this figure to Fig. 4(b) shows that more endmember candidates are found.

There are *more* virtual skewers per block when zero-valued coefficients are permitted, but those skewers are also closer to each other.

Restricting ourselves to corners of the hypercube, we get fewer virtual skewers per block ($O(2^B)$ instead of $O(3^B)$), but those skewers are more widely separated. For the same number of virtual skewers, we can use a larger block size B, and therefore cover a higher dimensional subspace of the full D-dimensional sphere with a single block.

The first derived dot product requires $B - 1$ additions (but with $B \ll D$, this is a lot cheaper than the D multiply-accumulate's required for a real dot product); furthermore, by organizing the order of computation of the derived skewers like a "graycode," it is possible to compute each of the remaining derived dot products with only a single addition; see Fig. 3(b).

3.3. Alternate corners

It is clear from the previous section that a goal in choosing directions for virtual skewers is to try and cover as much of the surface of the D dimensional unit sphere as possible. The alternate corners strategy considers a subset of the 2^B coefficient vectors $\mathbf{a} \in \{-1, 1\}^B$ which still "covers" the B dimensional space, but not as finely as the full set. One measure of the granularity of this coverage is the characteristic distance between coefficent vectors and their nearest neighbors in the B-dimensional coefficient vector space. For binary-valued coefficients, this is effectively a Hamming distance; and when the Hamming distance between a pair of $\mathbf{a}$'s is k, then the angle between the derived skewers is θ where

$$\cos\theta = 1 - \frac{2k}{B} \tag{8}$$

The case $k = 1$ produces all the corners of the hypercube; $k = 2$ produces only alternate corners, of which there are 2^{B-1} in a B-dimensional cube. For even-sized blocks $B \equiv 0 \pmod 2$, the corners on opposite ends of the cube are alternate corners; if we compute both min and max of every derived dot product, then the alternate corners will be taken care of, and we will effectively get 2^{B-2} derived dot products for each block of B skewers. (For odd-sized blocks $B \equiv 1 \pmod 2$, there is no advantage to taking alternate corners since computing both min and max of every dot product is effectively the same as using all corners.)

The advantage of having virtual skewers farther apart is illustrated in Fig. 5. The speedup increases less rapidly with B, but for a given speedup, more endmembers are identified.

In general, the problem of identifying a maximal set of vectors in a B-dimensional binary space such that every pair of vectors has a Hamming distance at least as large as a specified k is a classic problem in communication theory.

One would like to produce "codes" for which few-bit errors and be detected and/or corrected, and to do so requires sets of binary vectors with large pairwise Hamming distances. A "perfect" code satisfies

$$A(B,k) = \frac{2^B}{\sum_{i=0}^{(k-1)/2} \binom{B}{i} 2^i}, \tag{9}$$

but for for most values of B and k, this is an upper bound. In fact, for many values of B and k, the actual size of the maximal code is unknown. Nonetheless, algorithms exist for finding "good" codes, given values of B and k. For $k = 3$, Hamming found perfect codes whenever B is of the form $2^m - 1$; the number of code vectors in this case is given by 2^{2^m-1-m}, and so the gain is given by

$$\frac{2^{2^m-1-m}}{2^m - 1} \approx \frac{2^B}{B^2} \tag{10}$$

virtual skewers per actual skewer. (See Roman[14] for an accessible introduction to error-correction and error-detection coding.)

3.4. Orthogonally aligned skewers

In previous sections we have spoken of randomly chosen vectors $\mathbf{k}_b \in \mathcal{R}^D$; however, we can avoid even computing the original B dot products by choosing the vectors $\mathbf{k}$ appropriately. In particular, we consider choosing the $\mathbf{k}$'s from among the D axes of the original data. There are

$$\left(\begin{array}{c} D \\ B \end{array} \right) = \frac{D!}{B!(B-D)!} \tag{11}$$

ways to choose a block of B random othogonal skewers from the D orthogonal components, so for even moderately large D and B, we are not giving up a lot of randomness by limiting ourselves to orthogonal vectors.

3.5. Principal Components

An advantage of the PPI algorithm over some alternatives is that it works reasonably well in high-dimensional spaces; if the effective dimension of a data set is D', but the data are nominally expressed as D channels, then the only penalty for working in the nominal space instead of the reduced space is D/D'. For some algorithms, this penalty is exponential in D.

Nonetheless, there is still an advantage in working in a reduced space if one is available. And principal components analysis provides a relatively convenient way of projecting the data to a lower dimensional space while still maintaining most of the variability in the data. The principal components algorithm is not cheap, however, and the matrix factorization involved is very floating-point intensive, which makes a full-hardware implementation less attractive.

The first (and most expensive) step in a principal components analysis is the computation of the covariance matrix. This requires $O(ND^2)$ scalar multiply-accumulate operations, although shortcuts are feasable here (*e.g.*, use a sampling of $N' \ll N$ points to estimate the covariance matrix; or use the covariance matrix from a previous experiment). The matrix factorization, usually a singular value decomposition, does not have the O(N) pre-factor, but it does scale as $O(D^3)$, with a large (of order ten) coefficient. Having determined the D' largest principal components, one can rotate the data with $O(ND')$ dot products, each in the nominal D-dimensional space, for a total effort of $O(NDD')$ multiply-accumulates.

The standard PPI algorithm, with K skewers, requires $O(NDK)$ multiply-accumulates in the nominal space, and $O(ND'K)$ in the lower-dimensional space.

With $K \gg D$, the cost of rotating the data is small compared to the cost of the PPI, so the gain in going to principal components is the factor D'/D. This is only linear in the dimension, but for hyperspectral data with hundreds of channels, and projections to a few tens of principal components, gains of an order of magnitude are available.

A possibly greater gain, in combining principal components with the fast PPI algorithm, lies in the projection of the D-dimensional space to a lower B-dimensional space.

4. HARDWARE ACCELERATION

The pixel purity algorithm is a good candidate for acceleration because it is readily parallelizable (the skewer calculations are done independently) and the core calculation is a simple dot product. Hardware acceleration of a dot product is by itself useful, as it is the "inner loop" for many remote sensing and multispectral image processing tasks.[15]

This includes the N-FINDR[11] algorithm, which attempts to inscribe the largest-volume simplex in the data cloud. Here, the quality of a given data point as the potential replacement for a given fiducial vertex can be expressed as a dot product of that data point with the "skewer" that is perpendicular to the sub-simplex that does not contain the fiducial vertex. The computation of the skewer is somwhat involved, but it scales only with the dimension of the simplex, and not with the size of the data set. So for large data sets, the dominant computation is the dot product of very particular skewers with the full dataset.

We have designed an implementation of the standard PPI algorithm on reconfigurable hardware.[16] The basic architecture is shown in Fig. 6. It is composed of a matrix of dot-product operators. Each dot-product is fed serially with the D components of the pixels and the skewers; this requires D cycles, but in that time a dot product is computed for each operator in the matrix.

The results of the dot-product are stored into registers (shaded boxes) and shifted to the MinMax units. These units compute the minimum and the maximum of the dot-product and keep track of which pixel produced the extreme values. The results of the MinMax units are shifted to the host processor through a fifo.

In order to get the best performance, the minimum and maximum operations are performed in parallel with the dot-product computation: as soon as a dot-product phase is accomplished, the results are stored in the shift registers, and another phase begins immediately. During the next dot-product phase computation, the previous dot-product values are bit-serially shifted to the MinMax units. Hence, there is a complete overlap between the dot-product and the minimum/maximum computations.

One issue that arises in the hardware implementation is the precision of skewer coefficients. If the skewers are limited to a few bits of precision per coordinate direction, they can still provide an effective "cover" of the D-dimensional sphere of possible directions, but the dot product computation is faster (and more to the point, since it takes up less real estate on the reconfigurable chip, is more parallelizable). In our implementation, we use three bits (allowing the coefficient to vary from -3 to 3), but Fig. 7 shows the effect of reduced precision in the plot of cumulative endmember candidates. Reduced precision does indeed reduce the number of candidates, but the effect is small, which gives us confidence in this design choice.

We have started to investigate the hardware acceleration of the block-skewer approach, but this is currently a work in progress.

5. PPI AS A PRE-PROCESSOR

Since PPI produces candidate endmembers, we can use these as input to more sophisticated algorithms which can then be used to identify the best choice of endmembers. To illustrate the idea, we used PPI to produce a stream of endmember candidates, and then used these pixels as input to our implementation of the NFINDR[11] algorithm. NFINDR attempts to inscribe the largest volume simplex in the data and the effort to do so scales linearly with the data. (It is to the credit of the NFINDR authors that it is so fast – there is a combinatorially large number of possible simplices one might inscribe in a given data set.) Although NFINDR is quite efficient at processing data, some more sophisticated approaches, such as the archetypes of Cutler and Brieman,[4] require processing that scales very rapidly with the number of data points. If a fast PPI can be used to rapidly identify candidate endmembers, then these more sophisticated approaches can use that to speed up their efforts to identify the actual endmembers. In Fig. 8, we supply NFINDR with endmember candidates from PPI applied to ten-dimensional simulated MTI data, and we find that it is able to rapidly identify the largest volume simplex from this data. As a comparison, we supply NFINDR with random subsets of the pixels from the image, and again it finds the largest volume simplices that these subsets can support. What is most evident in this comparison is that the simplex volume computed with only a few of the PPI candidate endmembers is much larger than the volume computed with random subsets of much greater size.

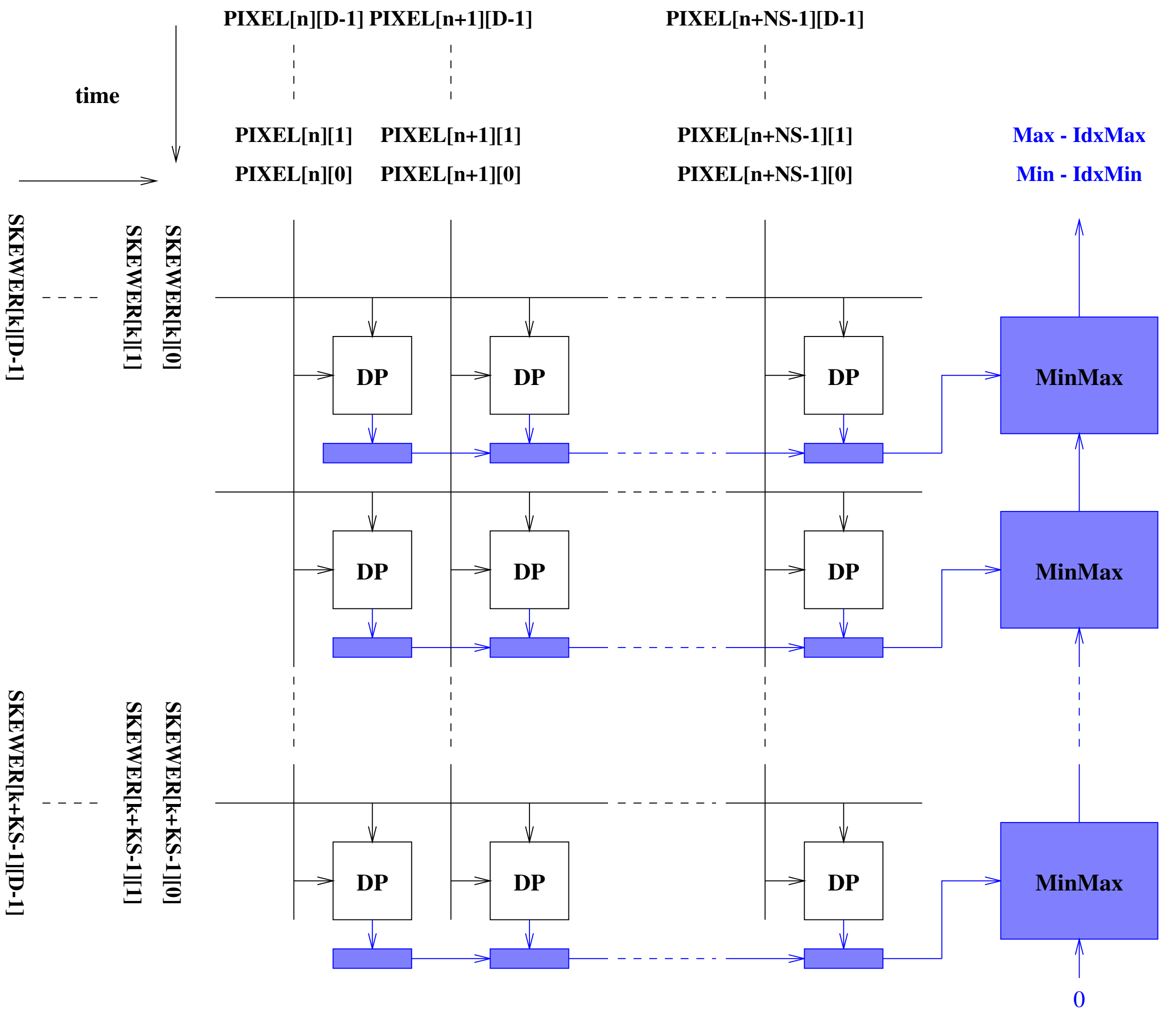

Figure 6. Basic architecture of the hardware implementation of the Pixel Purity Index algorithm. A matrix of dot-product operators (DP) operate in parallel, taking the components of the skewers on the horizontal axis and the components of the pixels on the vertical axis. The results are fed into MinMax units which identify the extreme dot products.

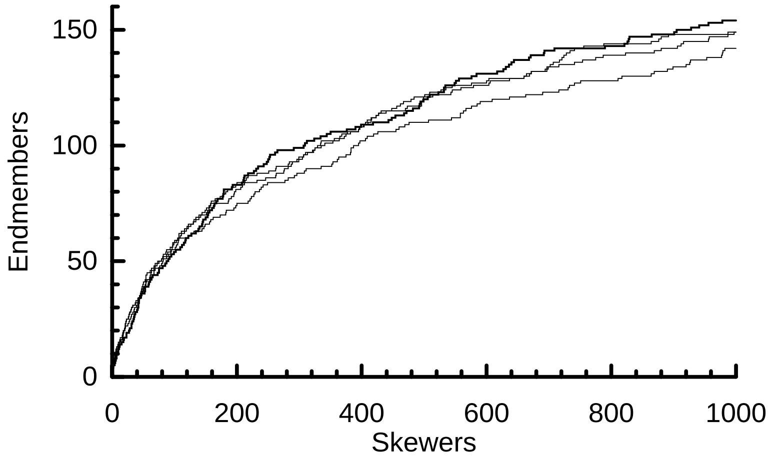

Figure 7. Cumulative plot of candidate endmembers as a function of the number of skewers is shown for different discretizations of the skewer coefficients. The darker line at the top is the standard PPI, and is the same as Fig. 2(a); the lighter lines correspond to skewer coefficients discretized to 2, 3, and 4 bits of precision. For example, the 3-bits case only permits the seven integer coefficients that vary from -3 to 3, inclusive. The 2-bits case is the lower curve, and in general, the more bits of precision that are used, the more candidate endmembers are found, but beyond two bits, the effect is not substantial.

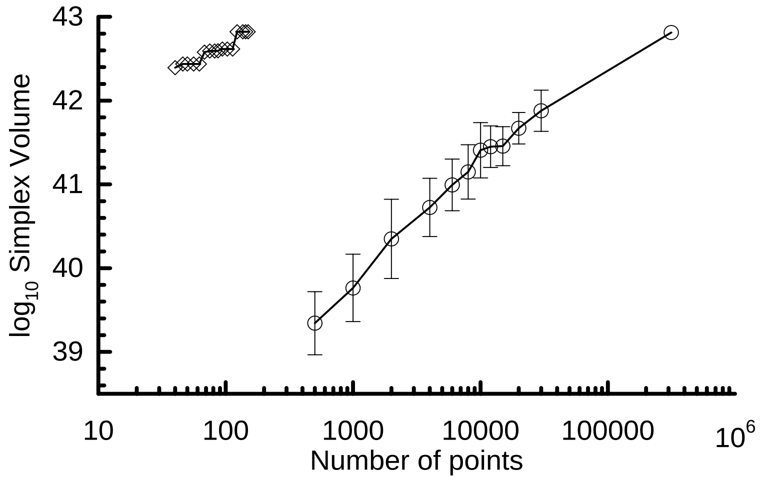

Figure 8. This plot shows the results of two separate experiments. In both experiments a number of pixels was supplied to NFINDR, and NFINDR produced the volume of the largest simplex supported by the pixels. The diamonds correspond to the first 150 candidate endmembers identified by PPI. The circles correspond to large but random subsets of the pixels in the image. This figure illustrates the effectiveness of PPI as a preprocessor.

ACKNOWLEDGEMENTS

We are pleased to acknowledge Chris Borel, for pointers to the literature on pixel purity, and for general discussions of endmembers and their applications. We also thank the Multispectral Thermal Imager team for supplying AVIRIS and simulated MTI data. Finally, we are grateful to the other members of the ASAPP team for many useful discussions.

REFERENCES

1. 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.
2. 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.
3. J. W. Boardman, "Geometric mixture analysis of imaging spectrometry data," *IEEE* , pp. 2369–2371, 1994.
4. A. Cutler and L. Breiman, "Archetypal analysis," *Technometrics* **36**, pp. 338–347, 1994.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. M. E. Winter, "N-FINDR: an algorithm for fast autonomous spectral end-member determination in hyperspectral data," *Proc. SPIE* **3753**, pp. 266–277, 1999.
12. NASA, 1999. `http://makalu.jpl.nasa.gov/avaris.html`.
13. 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.
14. S. Roman, *Introduction to Coding and Information Theory*, Springer, New York, 1997.
15. 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**, 1999.
16. D. Lavenier and J. Theiler, "FPGA implementation of the Pixel Purity Index algorithm for hyperspectral images," Tech. Rep. LA-UR-00-2426, Los Alamos National Laboratory, 2000.