Implementation of Wavelet Transform Image Compression Algorithms Using Associative Computing Based DSP Chips

Aviram Sariel

Associative Computing, Ltd., Ra'anana, Israel

Pankaj Das and William A. Pearlman

Electrical, Computer, and Systems Engineering Department

Rensselaer Polytechnic Institute, Troy, NY 12180-3590

Wavelet Transform is known to produce the most effective and computational efficient technique for image compression. The optimum space-spatial frequency localization property of this transform is utilized in the Embedded Zero Tree (EZW) Wavelet Coding which has been refined to produce best performance in SPIHT (set partitioning in hierarchical trees) algorithm for lossy compression and S+P (S-transform and prediction) for lossless compression. Using the multi-resolution property of wavelet transform one can also have progressive transmission for preliminary inspection where the criterion for progressiveness could be either fidelity or resolution. The three important points of wavelet based compression algorithms are:
(a) partial ordering of transformed magnitudes with order transmission using subset partitioning
(b) refinement bit transmission using ordered bit plane
(c) use of the self-similarity of the transform coefficients for different scales.
This paper deals with the implementation of these compression algorithms using Associative Computing DSP chips. Associative Computing performs search and modifies operations on very large blocks of data in parallel, making this technology particularly suitable for correlation and pattern matching. As the wavelet transform is based on correlation, this DSP chip has a natural advantage over others. It will be further shown that Associative Computing provides a natural VLSI architecture for the efficient implementation of SPIHT, enabling major speed-ups and a variety of system architectures.
The paper describes general issues of implementing wavelets\SPIHT image compression using Associative Computing machines and specific implementation details on ACL's XU chip, an Associative Computing chip with 2K SIMD processors and a word length of 128 bits. The XU enables a parallel processing of entire embedded trees, which allows for a one-pass image encoding, using S+P image transforms and SPIHT based coding methods.

1.1 Introduction to image compression:

In a broad sense, image compression is based on the fact that the numerical values of neighboring pixels are very closely correlated. Those similar numerical values can be used for compression via the following strategies:

a) Transform the original image, represented by a table or matrix, containing independent data for each pixel into a transform domain, which contain data about pixel groups. This will typically result in a transform representation with much smaller numerical values, which need fewer digits for representation. To allow perfect reconstruction of the original image, the transform needs to be perfectly reversible.
b) Send the image, or the transformed data, with an efficient protocol. The protocol transmits efficient commands to the decoder, which then reconstructs the original bit stream. For perfect reconstruction, the entire bit stream of decoder commands must be recovered.
c) Lossy compression is usually based on substituting the exact transformed data with data that allow reconstruction of an image similar to the original, but described with much fewer bits. The differences between the new and old images are the reconstruction errors. Another lossy coding approach was devised by Shapiro [1], and later improved by Said and Pearlman [2], using a unique transmission protocol, that will be detailed below.

We term the (a) type compression as *Source Coding*; this refer generally to differential coding, DCT and block DCT methods, and to wavelet transforms. Lossy schemes have different guises, which, for DCT and wavelet, classically include building a lossy representation via spatially tuned thresholds, equating the transform coefficients below the threshold to zero, and sending only non-zero coefficients of the approximation. This construction of approximate transform coefficients is termed *Quantization*.

We term the (b) type coding as *Channel Coding*, meaning that it should be viewed as an encoder-decoder communication protocol.

Most compression algorithms, and mainly JPEG, have a block diagram similar to the following:

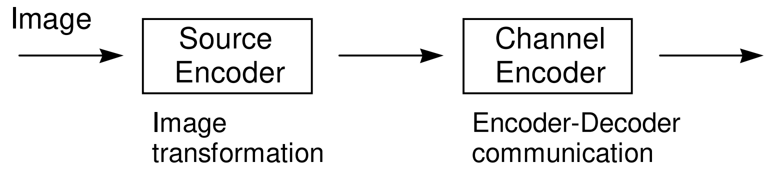


Fig. 1: Compression system algorithm block diagram

In the baseline JPEG, this takes the shape of

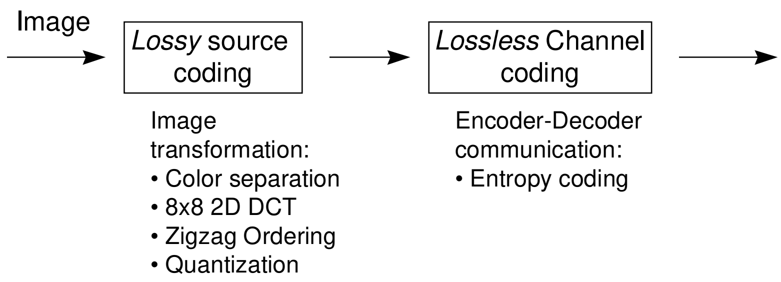


Fig. 2: Baseline JPEG

Wavelet compression is achieved by applying an adequate discrete wavelet transform (DWT) to the image, thus performing the decorrelation on many scales of the image.

One-dimensional (1D) DWT implies partitioning a 1D signal into two signals, where one contains the low pass frequency components of the original signal, and the other contain the high pass frequency components. Each of the new signals has exactly half the bandwidth of the original signal, thus being a highly redundant description of the original signal. This redundancy is removed by downsampling the low and high passes by 2, creating a critically sampled description of the original. In practice this amounts to performing each filter on either the even or odd components of the original signal. The process is then repeated on the low pass signal, and can be iterated to any depth to produce a set of octave size subbands of the original signal.

A two-dimensional (2D) DWT is usually implemented by using the same 1D filter pair on rows and columns sequentially, thus creating a separable 2D DWT transform. A block diagram of the separable 2D transform is illustrated in Fig. 3.a

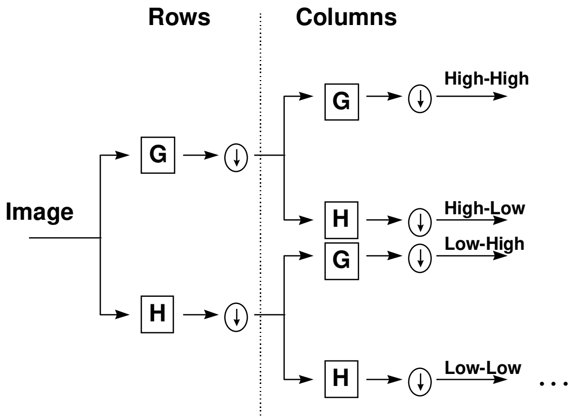


Fig 3.a: 2D wavelet transform via separable 1D transforms

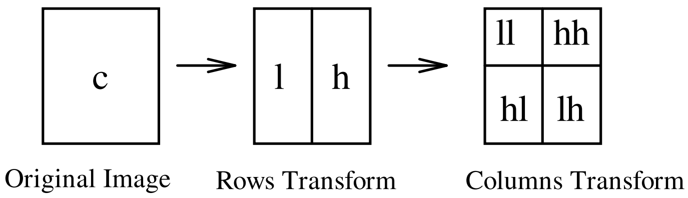


Fig 3.b

Fig 3.b shows the standard representation of one level of decomposition of a DWT transformed image. Note that the transform pyramid structure is used due to its multiresolution and self-similarity properties, but is basically only one specific choice of transform coefficients arrangement.

The arrangement described above is by no means unique. Separable transforms are easy to design, as they consist of a simple expansion of 1D wavelet theory, and can use available 1D wavelet filters. Since we are dealing in image representation, different filter pairs should be used for rows and columns, accounting for the higher horizontal correlation in, say, outdoor photos. Another design degree of freedom missed by the separable system is the option to use completely non-separable 2D filters, which deliver better 2D decorrelation (see Stanhill and Zeevi: [4], [5]).

Wavelet filters used in this context are identical to hierarchical subband system, with the subband logarithmically spaced in frequency and represent octave band decomposition. The filters are scaled such that the sum of the squares of the filter coefficients is 1. If the filters used are orthogonal wavelets, then the transform is unitary.

An extension of the DWT approach was introduced via integer to integer coding. Pixel values are integers (typically 8 bit), but transform coefficient are not necessarily so. Due to binary representation limitation, perfect reconstruction of non-integers is generally impossible on a digital machine. In this sense, the DCT transform is lossy even without the quantization; JPEG hardware implementation usually include defining some compromise between accurate fractional representation (better for reconstruction error minimization), processing rates and memory. Said and Pearlman [3] introduced a neat solution, by using a 1:1 integer to integer transform, thus achieving a truly lossless representation of the original signal. The 1D transform, termed S+P, is structured as follow:

low pass:

(1) $l[n] = \lfloor (c[2n] + c[2n+1]) / 2 \rfloor, \quad n = 0,1,\ldots, N/2 - 1$

high pass:

(2) $h[n] = c[2n] - c[2n+1], \quad n = 0,1,\ldots, N/2 - 1$

This defines the S transform. Said and Pearlman then use a predictor for the high pass:

(3) $h_d[n] = h[n] - \lfloor \hat{h}[n] + 1/2 \rfloor, \quad n = 0,1,\ldots,n/2-1$

Where ht[n] is defined by an estimation relation based on linear combinations of the previously defined low pass and high pass signals.

Define

(4) $\Delta l \equiv l[n-1] - l[n],$

The general form of the estimator is

(5) $\hat{h}[n] = \sum_{i=-L_0}^{L_1} \alpha_i \cdot \Delta l[n+i] \ - \ \sum_{j=1}^{H} \beta_j \cdot h[n+j]$

The filter is integer reversible (i.e. - truly perfectly reversible) by using (3). The prediction coefficients can be chosen either specifically per given image, using an optimization process, though this approach is extremely costly in execution time. Several "universal" predictor coefficient sets have been defined, each set applying to some class of images, such as medical, recreation, etc. Typical cases of coefficients are described in Fig. 4.

Parameter	Predictor		
	A	B	C
α_{-1}	0	0	-1/16
α_0	1/4	2/8	4/16
α_1	1/4	3/8	8/16
β_1	0	2/8	6/16

Fig 4: Prediction coefficient table (from Said & Pearlman [3])

Several notes on the S+P transform:

a) S+P is not linear, due to the rounding operations
b) The low pass can be represented with exactly the same number of bits as the original signal, and the high pass requires a well-defined bit number for perfect reconstruction. This provides a very nice feature to have constant processing memory for the entire transform.
c) Note that in contrast with JPEG, using S+P (or similar transform) imply a perfect representation of the image transform. The lossy part of the compression does not occur in the transform part (as is the case with DCT + quantization).

1.2 Significance Trees Quantization:

Wavelet compression channel coding is achieved via a scheme remarkably different from JPEG. Following Davis and Chawla [6], a unified approach termed Significance Trees Quantization (STQ) is presented. STQ is a generalization of the classic algorithms introduced by Shapiro[1], and Said & Pearlman[2]. Significance coding methods are based on partitioning the output of an image transform (typically wavelet) to hierarchical trees, where each transformation coefficient is either a node in the tree or a leaf. The trees are chosen so as to increase the probability that each node will have a value roughly equal (i.e. - having the same number of significant bits). The partitioning (see Fig. 5) usually follows a natural pyramid structure of wavelet transforms, with the DC coefficient in the tree root, the next (finer) level correction as root descendants, and each of them points to a finer level coefficients describing the same spatial location in the original image (see Fig. 6).

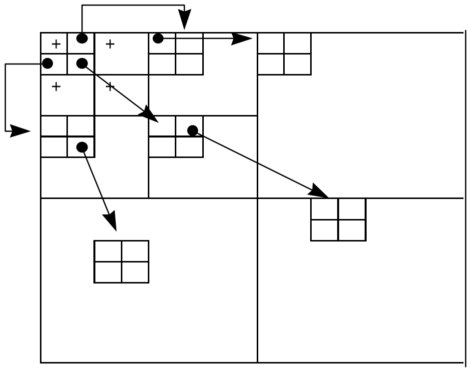
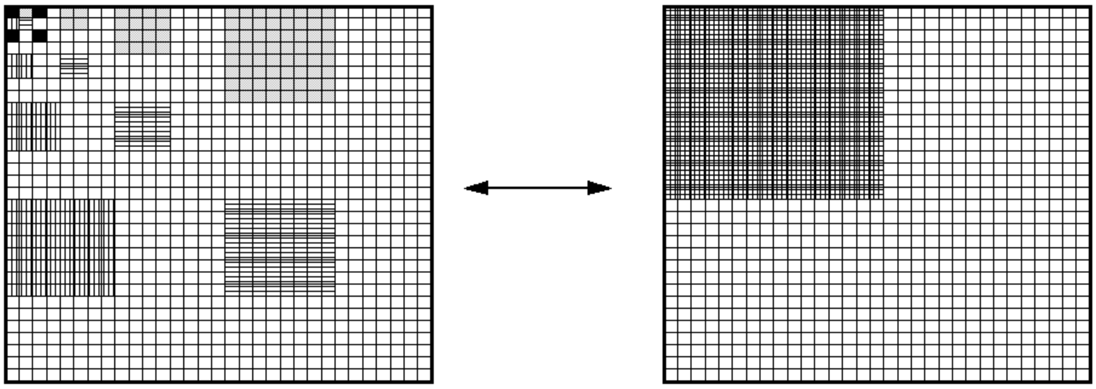


Fig 5: Standard tree structure



3 subtrees 1 spatial block

Fig 6: Trees as spatial location grouping

The STQ method then defines a transmission protocol that has the unique feature termed Embedded Coding. In this method, each bit transmitted directly decreases the MSE error in the reconstructed image. Embedded Coding allows a convenient encode-once-decode-many approach, useful for Internet applications, Digital Television Broadcast and so on, as well as exact bit rate control, a design target in many applications.

To achieve the embedded coding, STQ decoder assigns zero to all coefficients. The encoder now makes an initial pass on the data, and sends the MSB of the entire image (i.e. - 7 bits if all coefficients are in the range between -128 and 127). Note that the binary representation assumed here consists of a sign bit + absolute value bits, as opposed to a two's complement representation. Now the decoder makes a first information pass on the data, sending either '1" indicating a subtree - or a coefficient - with MSB bits (termed *significant*), and '0" to indicate a subtree where not even one coefficient with MSB bits exists (*insignificant*). A good choice of tree hierarchy should result in subtrees where coefficients with the same number of bits are packed together, allowing for a single '0" to describe the entire tree magnitude as 'smaller then MSB". After sending the magnitude description bits ("Magnitude" pass), STQ algorithms send the sign bits of the coefficients having MSB bits. The algorithm then proceeds to check the trees for N=MSB-1 bits, in a process identical to the one described above. After checking the new magnitudes and signs found for N bits, the encoder sends the Nth bit for each of coefficient discovered in the previous pass. This second pass is termed the 'Refinement" pass. The algorithm now proceeds, by sending the magnitude and sign bits for N=MSB-2, and Nth bit refinements for the coefficients found in previous passes. All the ordering data for the sent bits are defined by the magnitude transmissions, as they contain the exact subtrees paths that are used to arrive to a specific coefficient of N bits. The process continues until all bits are sent for all coefficients for lossless coding or the bit budget is exhausted for lossy coding. If the transform is unitary (via, say, orthogonal wavelets), then the MSE of the reconstructed transform is identical to the MSE of the reconstructed image (Euclidean norm being invariant under unitary transformation). Since the image MSE is quickly reduced by sending the transform most significant data first, this defines the compression mechanism as

a) arrange data by significance (i.e. - wavelet transform)

b) send data by significance (STQ)
Integer to integer transforms are not linear due to the rounding operation and hence can only be approximately unitary. However, finite digital representation does in effect produce only non-linear transforms due to truncation. The common heuristic supporting the use of the pseudo linear transforms is that it produces results close to analytic computation, and a similar heuristic can be observed to hold for the S+P transform (i.e. - image reconstruction MSE is near the transform reconstruction MSE), and the mechanism remains (provided the S+P output is scaled through bit shifts to produce an approximately unitary transform).

The description of STQ as transmission protocol is derived from EZW and SPIHT, and is open to implementation. Both EZW and SPIHT were originally presented solely in the form of a preferred implementation, using linked lists as an abstract data structure.

In the remainder of the paper we discuss a remarkably different approach to STQ implementation on a massively parallel chip. It can be used for EZW coding, or image-optimized STQ coding of a kind; the approach to be presented resembles, if any, classic Multigrid PDE numerical solvers, and hence sets SPIHT naturally as a multiresolution representation of the transform. Implementation details are divided in two: first the general approach to massively parallel implementation, and then the actual hardware implementation using active associative arrays developed by Associative Computing Ltd.

2.1 STQ block processing:

The coding implementation system diagram is presented in Fig 7.

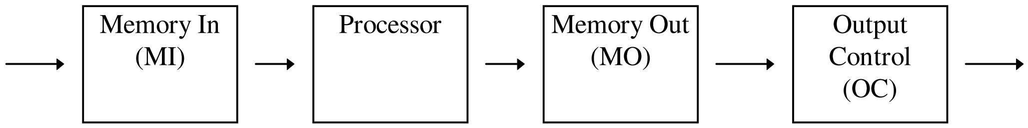


Fig. 7: Coding System Model

Adding the MO and OC unit is almost compulsory, as processors rarely interface directly to communication. Those devices can be used for SPIHT block processing: consider performing SPIHT, or other STQ, on a specific subtree in the transformed image, and then writing the resultant bitstream to MO in a single row (the MO being an abstract matrix); the written row will posses the embedded coding property, and hence its reconstruction error will decrease as we move along columns to its end. Writing all subtrees into MO rows will result in an entire 2D array having this property for each row. Accessing the array by columns for transmission sends the data in bit packets whose size corresponds to the number of subtrees in the image root; those packets still keep the embedded coding property! Furthermore - “sliding” a specific subtree row towards the end is equivalent to reducing its reconstruction importance, thus creating a simple mechanism for spatial importance hierarchy, allowing to reduce the bit rate for less important regions in the image, while at the same time assigning the freed bandwidth to more critical regions, according to their position in the 2D array. We emphasis that this method should not apply to images alone - it is quite possible to use it as an adaptive lossy bandwidth allocation scheme for, say, multi-voice user line-doubling. The only things required are the embedded coding property and an efficient signaling protocol for the subtree start location. As can be seen, those locations are needed anyway, and are related to signaling each subtree MSB, needed for the reconstruction.

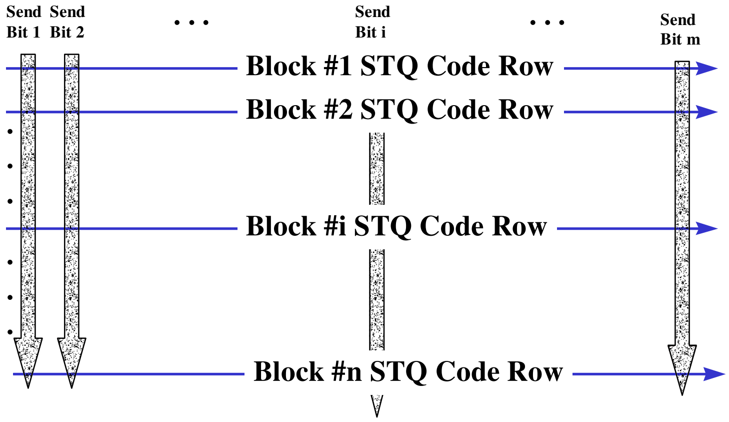


Fig 8: Encode rows, Transmit columns

2.2 Fast STQ algorithm:

The next problem to be considered is the computation of a single subtree STQ bitstream. Consider the standard tree structure described above in Fig. 5. For every leaf, it is possible to define a local STQ (i.e. - SPIHT\EZW) transform, which will deliver the leaf value in the form of [0,0,...,1(msb+1 location),sign, value bits]. The zeros being “insignificant” magnitude declarations for N >msb(Xj) (the current bit plane greater than the most significant bit of a coefficient Xj), then “1” for declaring the leaf as significant, then the rest of the leaf bits. This structure holds the embedded coding property - if we use subtrees composed of a single leaf, it is a valid (inefficient) STQ code.

Coefficients can be grouped to produced a combined STQ code using a common group descriptor, equivalent to nodes in the tree level immediately above the leafs. Each coefficient will be encoded locally as defined above, and the group descriptor (i.e. - node) will account for the common group properties in the form **[0,0,....,1(group max msb location +1), 1,1,1,1...]**. This in effect defines a transmission mask for the local coefficients, as the encoder can transmit the common descriptor zeros alone, then a single “1”, and from then emit only the local coefficient STQ bitstream for the leafs.

A more refined descriptor structure will have, as before, zeros for the group insignificant bit positions, then “1” for the group max msb, zeros again, then “1” for the group second largest value msb, zeros again, “1” for the group third largest msb, and so on, until end of description is reached. The result should look like **[0,0,...,1,0,0,...,1,0,0,...,EOD]**. This allow to transmit the mask zeros until the first significant value is reached, transmit this value local code bit, then move on to the transmit descriptor zero bit (= insignificant pass on remaining coefficients), then the next bit of the significant value local code (= refinement pass), and so on. If another significant value is discovered, it is transmitted with previous significant values. Now, note that the descriptor can be created by injecting the local STQ msb’s (located one bit ahead of the value msb) into a zeros string of appropriate length (entire subtree max msb). The main point of the description above is this: the descriptors can be defined based solely on the local STQ descriptions of the group.

Actual nodes in an STQ scheme differ from the above since the nodes are not pure group descriptors, but actual data members containing coefficients values. SPIHT solves this rather compactly, assigning after each “1” in the description string another bit, specifying whether the new significant value declaration corresponds to leafs or descendant nodes (“0”) or also to the node coefficient itself (“1”). After the first emission of such “1” (i.e. - node coefficient is significant), there is no need for further such information, and the descriptor continues as described above. The encoder will include the node coefficient in the refinement transmission and send it according to its order in the significant value list.

Another variation is the inclusion of sign bits, referring to the sign of the coefficient. Including the signs in the magnitude description follows the SPIHT method, but is usually redundant (i.e. - the descriptor sign bit will be emitted only if the magnitude bit is “1” and the node bit is “1” - the sign is essentially local information).

Iterating the procedure described is simple - higher (closer to root) node coding is done by injecting the lower node codes msb locations into a zeros string of appropriate length, adding sign and node magnitude bits after the 1’s in the description string, stopping the node bit addition after the node magnitude was delivered (i.e. - all node magnitude bits will be “0” except the last, after which no magnitude bits are added to the string).

Note that the nodes description maximal length is three times the MSB of the entire subtree; the node descriptors can be implemented as such 3MSB bits string, and high nodes description string of the same length can be generated from groups of lower nodes descriptions + the high node coefficient value. We define the Descriptor string according to Fig. 9: the description "string" is composed from three types of binary notations: ***mgn***, ***sgn*** and ***nod***. *Mgn* is a bit specifying if the descriptor and its descendants contain a coefficient of the relevant magnitude (each *mgn* entry corresponds to one bit plane pass). *Sgn* specifies the current coefficient sign, provided the magnitude is significant. *Nod* is a bit specifying if the previous *mgn* referred to the also to the node ("1") or to its descendants only ("0"). Note that if *mgn*=0, the sgn and *nod* data should not be transmitted (see later on).

[mgn | sgn | nod | mgn | sgn | nod | ..... | mgn | sgn | nod]

Fig 9: Description string structure

Starting from the leafs and working to the subtree root, the nodes deliver information upwards. For SPIHT and EZW, the path followed is identical to standard wavelet subband decomposition, and is based on the same pixel values. The computation of description strings can be coupled with the low pass part of the transform, as it is based on the same data. As the magnitude pass is composed of the description strings, the computation is made with only one pass on the subtree data.

For actual transmission, the description strings should be embedded with the local codes. Following the same notions as before, we consider this process as providing an adequate transmission mask for each bit in the description and value strings. When such a mask is supplied, the transmission will include passing on all subtree MSB description string bits (the strings MSB bit), transmitting only the unmasked ones (magnitude pass), then passing on all values, again transmitting their unmasked MSB bit, (refinement pass), then returning to the descriptors for the MSB-1 bit, and so on. If the strings and masks are arranged as 2D bits arrays, this amounts to sending the columns of the bitwise AND operation between the data arrays (descriptors and values) and their respective masks.

Using the notation of Fig. 7, the STQ encoder-decoder protocol is decomposed into two communication protocols: the first is between the processing device and its output control (where the output control emit only relevant bits, dropping - say - the *nod* and *sgn* bits when *mgn*=0). The second is between the output control and the decoder, which receives an STQ bit stream according to some specific tree scanning method, specific S+P transform, and so on. The decoder does not need to reconstruct the description strings, but only to apply the STQ decoding procedure (see Said and Pearlman [2]).

The first stage in the masking computation is to create masks for each description string. Note that each node can induce such a mask at its immediate descendants, as the lower descriptors *mgn* bits are relevant only if the parent node *mgn* bit is 1. *Sgn* and *nod* bits in the lower descriptions need to be transmitted only if the preceding *mgn* bit is 1, so the *Sgn* and *nod* bits mask is the AND of preceding *mgn* bit mask with the preceding *mgn* bit itself. Iterating towards the leafs results in a complete masking of the subtree descriptors, i.e. complete making for magnitude passes. Magnitude masks are binary decisions transferred along the tree from top to bottom; their computation requires full solution of the description strings.

Masks for the value strings, which compose the refinement passes, are computed locally. A value bit is transmitted if the value was found significant in previous magnitude pass, and hence each coefficient will be valid for refinements from its msb bit downwards. If the sign is transmitted during the magnitude pass this turns into validity from msb-1 downwards. Value bit masks computation can therefore be coupled with any coefficient computation; if not, they can be achieved in any pass on the transformed image.

To summarize, Magnitude passes are computed up, then down. Refinement passes are computed locally, and integration between passes is achieved by sending the relevant bit plane of the magnitude (unmasked magnitude bits), then a relevant refinement bit plane, and so on. The computations include sending information up the tree, and decisions downwards.

Formally:

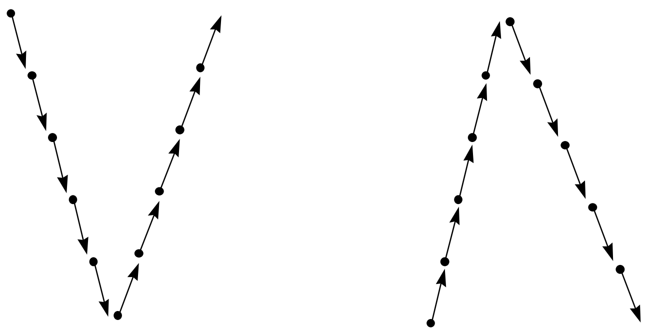
A. Transform image Img(i,j) using a suitable wavelet integer to integer transform, denoting the transform coefficients C(i,j)
B. Define TMBR - Transform Maximal Bit Representation, accounting for the maximal possible number of bits required to represent a coefficient. Note that TMBR is constant for given image bit depth and given transform.
C. Define TABR - Transform Actual Bit Representation = max(msb(C(i,j))) + 1, being the actual maximal number of bits required to represent all signed transform coefficients. In both EZW and SPIHT the TABR requires an independent computation pass. It can be computed during the transform, and sent as transform header.

D. Partition C into subtrees, defined by reversing the subband process (i.e. - the classic subtrees). Denote O(i,j) the set of offspring coefficients for coefficient (i,j) (i.e. - immediate descendants). Denote V(n) the set off (i,j) coordinates of pixels in level n of the tree (n=0 being leafs)
E. compute coefficients Refinement binary mask RM(i,j) {RM(i,j,t) = 1 for t<msb(C(i,j))-1, 0 else; t=1,... TMBR}
F. Calculate local magnitude descriptors MD(i,j) on the node and leaf levels : {MD(i,j,3*msb(C(i,j)))=1, MD(i,j,3*msb(C(i,j))-1)=Sign(C(i,j)), MD(i,j,t)=0 else; t=1...3*TABR}
G. Set n=1, and start working on nodes
H. Add node self magnitude information: {MD(i,j, 3*msb(C(i,j))-2)=1; (i,j) $\in$ V(n) }
I. Update Node descriptor: {MD(i,j,t)=AND(MD(i,j,t), MD(O(i,j))); C(i,j) not leaf} (i.e. - AND between the node descriptor and all offspring descriptors)
J. Set n=n+1 (while n < number of subtree levels), and return to (H)

Now the Magnitude descriptions are computed, start magnitude mask. The mask consists of a 2D array, of exactly the same dimensions as the magnitude description array.

A. Start at the root: assign “1” for every **mgn** mask bit of the root
B. Move one tree level down, into descendant set. On each descendants, perform:
C. Set **mgn** bit mask same as parent **mgn** bit
D. If current entry is node, set entry **nod** mask bit to AND(**mgn**, **mgn** mask bit); if entry is leaf, set entry **nod** mask to “0” (i.e. - do nothing)
E. Set entry **sgn** bit mask to **mgn**
F. If not on tree leaf level, return to (B)

The scheme is presented in Fig 10:



MultiGrid V Cycle **Fast STQ L Cycle**

Fig 10: Fast STQ L Cycle

Fast STQ flow clearly resembles the flow of multigrid algorithms known as “V cycle” (see Brandt, [7], and Briggs, [8]). Due to exactly opposite definitions of trees “roots” and “tops”, we move from multigrid V cycle to fast STQ Λ cycle (Fig. 10); the descriptions, however, both retain the idea of delivering information from fine grain scale to gross grain scale, and some sort of decisions (transmission masks in our case, and smoothing corrections for the PDE multigrid solvers) going from gross scale to the fine.

It should be stressed that the Fast STQ scheme presented above is not dependent on a specific tree choice, on specific definition of the significance function (Said & Pearlman [2] notation), provided that both (tree structure and significance function definition) are known apriori to the decoder.

3.1 Associative Processing Chip:

Associative processing is a non-standard method of designing and using digital machines. Instead of conventional CPU-memory duality, which inevitably leads to communication bottlenecks, the associative processor is composed of some several thousands “smart” memory cells, which can be written and read by content (hence the “associative” terminology). As shall be seen, those two operations (+ a third) allow full programmability of the memory cells array, and it can therefore should be viewed as a massively parallel processors array, working in SIMD (single instruction multiple data).

Fig. 8 presents a common abstraction for associative processor, termed *Associative Processing Array (APA)*:

1) *Array*, composed of several thousands of memory units, each of 128-256 bit size.
2) Response register, termed *Tag*, abstractly positioned as a column to the right of the array.
3) *Pattern* and *Mask* registers, which allow for associative transmission of read\write commands.

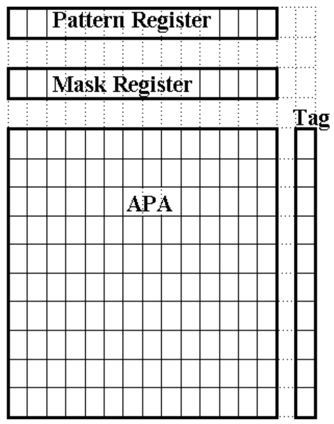
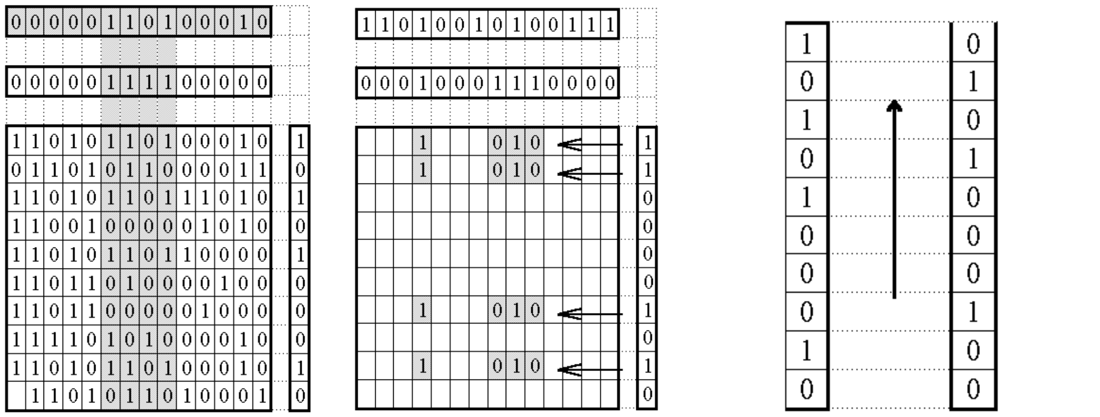


Fig 11: Associative array: array, tag, mask, pattern

Reading data from the APA is done with a *Compare* command, depicted in Fig. 12(a). A binary string is read into the pattern register, the mask register is set to mask the relevant part of the string, and the array then responds by assigning each of the bits in the tag register according to the result of the binary match performed on all masked array columns.

Writing data is achieved by using a *Write* command, as shown in Fig. 12(b). Pattern and mask registers are set, thus defining the relevant columns into which a pattern shall be written. A relevant bit pattern is in built into the tag, and then the write command is applied to input the pattern string into array columns masked by the mask register, and array rows masked by the tag register.

A third basic command (operand), used for inter-cell communication, is the *Shift* operand. The tag register can be shifted up or down, thus allowing 1D neighborhood operations. Results from a compare cycle can be shifted to be used as inputs to a write cycle.



a) ***Compare***: shaded cells matched against the pattern. Match is written into the Tag (one clock cycle)

b) ***Write***: shaded cells are written into (one clock cycle)

c) ***Shift***: original *Tag* (on the right) shifts one cell up (on the left). *Shifts* of other sizes are also available

Fig 12: *Write*, *Compare* and *Shift* operands

Further properties of APA based chips includes an I\O mechanism. In ACL's Xium-2, the mechanism is based on assigning a special part of the array columns (abstractly located on the array right end) , which can be used for array I\O. The special region is scanned sequentially from top to bottom (regular jumps can be defined), data input\output can be achieved in the same clock cycle, while concurrently performing any other array operations (i.e. - parallel I\O, and parallel-to-processing I\O).

Associative chip architectures can be readily defined based on the basic design. A simple upgrade path relates to the importance of the satellite registers: more tags can be added (Sherson, [9]), responders counter can be added to the tag (Krikellis and Weems, [10]), and so on. Regarding upgrades to basic APA designs, one cannot overemphasis the role of I\O to the array. The array is parallel indeed, but the complete system also includes external raw data memory, and processed data memory; thus are accessed sequentially,

3.2 Associative computations:

following is a pseudo code presentation of associative operators relevant to implementing associative wavelets and STQ. The *Settag* operand accounts for masking the entire tag register (i.e. - all bits equal 1).

1) ***OR***:
   c=OR(a,b) -

   pattern(ones); mask©; settag; write; // 1st clock cycle: assign 1's to output bit column

   pattern(zeros); mask(a,b); compare; // 2nd clock cycle: retrieve all cases where a=b=0. Those cells are now marked in the *Tag* register

   pattern(zeros); mask©; write; // 3rd clock cycle: assign 0's to all c cells where a=b=0

2) ***AND***:
   c=AND(a,b) -

   pattern(zeros); mask©; settag; write; // 1st clock cycle: assign 0's to output bit column

   pattern(ones); mask(a,b); compare; // 2nd clock cycle: retrieve all cases where a=b=1. Those cells are now marked in the *Tag* register

   pattern(ones); mask©; write; // 3rd clock cycle: assign 1's to all c cells where a=b=1.

3) ***NOT***:
   b=NOT(a) -

   pattern(zeros); mask(b); setteg; write; // 1st clock cycle: assigning 0's to b

   pattern(zeros); mask(a); settag; compare; // 2nd clock cycle: retrieve a=0 enties

   pattern(ones); mask(b); settag; write; // 3rd clock cycle: assign 1's to b cells where a=0

4) ***ADD***:
   d=a+b

   implement a full adder truth table: given to input number columns, a and b, and a result number columns d, the columns are indexed a_i, b_i, d_i, specifying bits starting with the lsb and moving on. A carry column c (one bit) is allocated, and resultant truth table is -

a_i	b_i	c_{old}	d_i	c_{new}
1	0	0	1	0
1	1	0	0	1
0	0	0	0	0
0	1	0	1	0
1	0	1	0	1
1	1	1	1	1
0	0	1	1	0
0	1	1	0	1

Fig. 13: Associative *ADD* truth table

Basic pseudo code implementation: apply *compare* to each input combination, followed by *write* of the appropriate output combination. This results in a 16 clock cycles/bit adder; better algorithms and system architecture are readily available: on ACL Xium-2 adding is achieved in 5 cycles/bit, and on next generation chip (currently termed XU), adding performance will reach 3 cycles/bit.

5) Other integer arithmetic follows directly from adding: multiplication by adding iteration, powers by multiplication iteration, and general functions via powers, multiplications and additions, or direct lookup tables. Rounding operations are done via arithmetically shifting the entire vector columns to the right, or by simply redefining the vector location (i.e. - addressing only relevant columns in subsequent operations).

6) ***1D Filters*** are implemented by use of the *Shift* operand:
   a) create an up-shifted copy of the vector:
      pattern(zeros); settag; mask (copy location); write; // clears copy area

      for i=1:msb

   pattern(original[i]=1); mask(original[i]); compare; shift(1 up); // Tag now holds a shifted up version of the original i bit column (2 clock cycles)

   pattern(copy[i]=1); mask(copy[i]); write; // writes shifted bit column to place

   end

   b) filter:
      *Output = Original + scalar*Copy*

   c) iteration:
      Iterating a-b (shifting the copy further up) allow simple implementation for any 1D filter.

It should be stressed again that the algorithms above were chosen mainly for their illustrative qualities. Detailed and lucid accounts of efficient algorithms and associative architectures are available at Akerib [11] , Sherson [9].

4.1 Associative Implementation of Wavelet Image Compression:

Associative (or any) computation of image wavelet transformation is composed of two main stages: 1) *arithmetic transformation*, accounting for all filter multiplications and additions and 2) *geometric transformation*, accounting for the rearranging the transformed data in some canonical pattern. For instance, the Lazy Transform is composed entirely of stage (2), as it includes partitioning the data to its odd and even members (it is a valid wavelet, and can be constructed as any 2D filter).

Without detailing the exact steps of ACL wavelet computation algorithms, we can still note that the arithmetic stage can be done rather cheaply, based on the 1D filter mechanism described above. 2D filters can be implemented as two repetitive 1D filters, where the (say) second pass will include long shifts to account for entire column (or row) alignment. The geometric transform part, if not properly designed, can consume many clock cycles if done using pure associative commands (i.e.

shifts). However, as the canonical representation geometric transform is identical for all wavelet filters, the following solution strategies can be formulated:

a) use serial memory control to place each outputted data to its proper place in an outer memory, then send the data from the memory by canonical ordering.
b) build a canonical ordering data outputting mechanism into the chip I\O , or better - build a random access I\O mechanism, and output directly in canonical ordering.
c) input the data in such an order that the output will automatically result in canonical order.

As the trade-off is between added memory complexity and added logic complexity, one should vote for larger outer memory to use an off-the-shelf chip, and probably vote for enhanced logic when designing a one-chip solution. Note that every solution will imply that the actual system architecture would rely heavily on the essentially serial I\O for the geometric transforms, and hence the system architecture should include those devices explicitly.

Associative fast STQ computation also necessitates a smart serial I\O mechanism. Implementation details are as follows:

1) input the transform coefficients into the array, ordered such that every node is above all its descendants, where the descendants are ordered in the same way for all the nodes.
2) transform coefficients to a [sign|abs(value)] representation
3) find subtree MSB (*compare* for zero columns until no match is found)
4) build **refinement** passes mask for each coefficient
5) start with the leaves - set **current nodes** = leaves
6) *shift up* all current nodes to align with their parent nodes (i.e. - make a copy of each descendant node in its parent node array row)
7) build parent **magnitude** descriptor
8) if not at tree root, set **current nodes** = parents (**current nodes**) and return to (4)
9) now at tree root, start moving down:
10) *shift down* each current node to align with its descendants (i.e. - make a copy of each current node in the same array row as its immediate descendants)
11) define descendants **magnitude** passes mask
12) if not on leaves level, set **current nodes** = child (**current nodes**) and return to (10)

Now the masks are set. Transmission should be in 3 bit columns for magnitude passes: the serial I\O reads each 3bit, and decides whether to send its first bit only (i.e. - "011" or "101". "011" is a "no significant coefficients" code, "101" is "significant descendant, pure node" code), send first two (i.e. - "100" declaring that the sub-subtree contain significant coefficients, but the node is not one of them), or send all three (i.e. - "111" or "110", declaring that the node itself is significant in the first two bits, and defining the node coefficient sign in the third bit). "000" stands for "don't send". Note that the dictionary is defined between the array and its serial gateway, and includes five words. The "101" symbol will emit "1" to the magnitude bitstream, which is interpreted correctly by the decoder since it has already decoded a previous significant coefficient at the node position.

After the 3 bit magnitude pass transmission, the array sends a 2 bit column for the refinement pass: first bit is a mask for the second bit (i.e. - for "10" or "11" emit "1", for "00" emit 0. The "01"'s do not happen).

Sending array columns out can be achieved either by some special column outputting mechanism in the I\O area, but this is expensive, requiring some 2K bit busses. In ACL Xium-2 there is an option for outputting the tags at 8 bit per clock cycle, which can be fine for the refinement passes 2 bit needs. Using pure associative commands, one can *Shift* up the current magnitude pass column to group together 3 bit descriptors in, say, 30 bit groups, creating a 30 bit columns which are accessed by the output control in 10 shifts jumps. The outputting can then be done in 30 bit (actually, 32 bit) buss, in rates close to 30 bits/cycle. Refinement passes can be sent similarly, in 32 bits/cycle.

4.2 Performance:

Associative performance evaluation is based on taking the maximum of two concurrent processes: I\O and processing. With the Xium-2 DSP chip, I\O is at 33 MHz , 32 bit in every clock cycle. A two-chip configuration (available as the Elan-2 board), will be used as a platform for processing performance. Assuming no delays at the serial output control, the processing cost is the combination of the above actions. As the processing cost is maximal when the cross-processor operations are at maximal distance, i.e. - the subtree is long, we shall calculate for subtrees which fill the length of the entire 2 chip array - 4K coefficients. This amounts to a 6 level image decomposition, with 1365 coefficients in each subtree of a 64x64 image region + the non-participating left-upper-corner coefficient (marked with "+" in the Fig. 5). Note that the three subtrees can be inputted to the array together, by properly assigning each subtree consistent bit locations in the 32 bit input process. To simplify the analysis, one can easily see that operations, which are performed in the same processor, can be completed in just a few machine cycles. Operations (2),(4),(7),(11) can therefor be neglected, as they include some combinations of the associative logical operators defined above.

Operation (3), which in serial processors leads to a pass on the entire subtree, is performed in the Xium-2 (or similar architectures) with just a few machine cycles: *Compare* the coefficients to an "expanding" pure zeros pattern starting from the MSB towards the LSB (not including the sign bits in the new representation), and count the number of responders in the *Tag*. When this number is non-zero, the subtree maximal representation bits are reached. Note that since this is a cheap operation, the protocol can be altered to differentiate between parts of the subtree, and perform this check on each separately.

Operations (6) and (10) are clock count heavy. Due to the symmetry between them, we shall analyze (6), and the reader can verify the result for (10). In each iteration of (1) - (12), operation (6) sends up the tree (= up the array) the coefficients from the previous level, aligning them with their parent node. The upward transportation applies only to **mgn** in the descendant's description strings. The trip duration for a single **mgn** column in the array is $4 \cdot 1 = 4$ in leaves level, $3(1+4\cdot 1)+1=16$ in the next level, $3(1+4(1+4\cdot 1))+1=64$ in the next, and so on: 4,16,64,256,1024,4096. In the Xium-2, the largest *Shift* is 16,which fits nicely. The clock cycles required for the shifts are therefor 2 (2 is also a valid *Shift*), 1, 4, 16, 64, 256. The total clock cycles needed for one **mgn** bit upward transmission are hence 343. As in the S+P transform the maximal number of bits is 10 (for an 8 bit input image), the number of clock cycles is 3430 at most (this can be further reduced, as only one of the three subtrees will need 10 bit representation). Operation (10), being symmetric, requires the same clock cycles, hence the total processing time is 6860 + low count operations. One can safely assume that the entire process is lower then 8K operations (i.e. - that low count operations consume less then 1332 clock cycles). The processing - which is performed on the three complete subtrees at once, is twice the I\O requirements, and hence both input and output can be completed during the processing. Output requires the serial output control to receive 5 bit columns in each bit pass (3 for the magnitude, 2 for refinement). Those can be packed into 30 separate $10 \cdot 3 = 30$ bit pack, and $16 \cdot 2 = 32$ bit packs, respectively.

Each bit pass output will consume $\lceil 4096/10 \rceil + \lceil 4096/16 \rceil = 666$ clock cycles, and for the entire 10 bit transmission - 6660 clock cycles. The total processing (input included) + output time is hence $8192 + 6660 = 14852$ clock cycles. For a 33 MHz machine, and 512x512 8 bit image, the throughput is 34 fps - > higher then real time.

It should be noted that shorter trees (i.e. 5 level decomposition) will reduce the processing of operation (6) and (10) to 870 clock cycles each, and keeping processing lower then input, and improving the throughput to 48 fps. Parallel I\O (i.e. - input while output), will further reduce the entire processing to the output 6660 clock cycles per region, which allow for 77 fps - much higher than real time.

5. Conclusions:

1) we have shown that STQ schemes, such as SPIHT and EZW, can be easily distributed or block processed, with the aid of memory buffer between the subtree encoder and the actual transmission control. The block processing methodology can be also used for assigning priorities for different image regions.
2) linked lists implementations for STQ's, which are typically time consuming, can be replaced by a Fast STQ scheme, which closely resembles MultiGrid V cycle algorithms, while still preserving the efficient encoder <=> decoder communication protocol, which is STQ main feature.
3) a preferred implementation for the subtree STQ coding calculation is an active associative array, such as the DSP chips developed by ACL. The associative approach allow fast filter performance plus a natural VLSI architecture for Fast STQ, which can be processed in rates much faster than real time.

References:

1) J. Shapiro, *Embedded Image Coding using Zerotrees of Wavelet Coefficients*, IEEE Transactions on Signal Processing, 41 (12), April 1993, pp. 3445-3462

2) A. Said, W. Pearlman, *A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees*; IEEE Transactions on Circuits and Systems for Video Technology, 6 (3), June 1996, pp. 243-249
3) A. Said, W. Pearlman, *An Image Multiresolution Representation for Lossless and Lossy Compression,* IEEE Transactions on Image Processing, 5(9), 1996, pp. - 1303-1310
4) D. Stanhill, Y. Y. Zeevi, *Two-Dimensional Orthogonal Filter-Banks and Wavelets with Linear Phase*; IEEE Trans. Signal Processing, 46 (1), January 1998, pp. 183-190. (CC Pub. No. 111, June 1995), Sept. 1997. (In Press).
5) D. Stanhill, Y. Y. Zeevi, *Two-Dimensional Orthogonal Wavelets with Vanishing Moments*; Trans. IEEE Signal Processing, 44 (9), October 1996, pp. 2579-2590
6) G. Davis and S. Chawla, *Image Coding Using Optimized Significance Tree Quantization*; Proc. Data Compression Conference, Snowbird, Utah, 1997.
7) A. Brandt, *Multi-Level Adaptive Solutions to Boundary-Value Problems*, Mathematics of Computation, 31 (138), April 1977, pp. 333-390
8) W.L Briggs, *A Multigrid Tutorial*, SIAM, 1987
9) I. Sherson, S. Ilgen, *A Reconfigurable Fully Parallel Associative Processor*; Journal of Parallel and Distributed Computing, Vol. 6, 1989, pp. 69-89
10) A. Krikelis, C.C Weems, *Associative Processing and Processors*; COMPUTER, Nov. 1994, pp 12-17, IEEE Computer Society
11) A. Akerib, *Associative Real Time Vision Machine*; Phd. Thesis, The Weizmann Institute, 1992.