ISSN No. 0976-5697

Volume 3, No. 4, July- August 2012

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

A Memory Efficient Listless SPECK (MLSK) Image Compression Algorithm for Low Memory Applications

N. R. Kidwai*
Department of Electronics & Communication Engineering,
Integral University, Lucknow, India
naimkidwai@gmail.com

Ekram Khan
Department of Electronics Engineering, Aligarh Muslim University, Aligarh, India
ekhan67@gmail,com

Rizwan Beg
Department of Computer Science and Engineering,
Integral University, India
rizwanbeg@gmail.com

Abstract: Use of data dependent lists to store state information during coding, SPECK (set partitioned embedded block coding) image coding algorithm requires large run-time memory, and thereby making SPECK image coder unsuitable for memory constrained applications. In this paper, a memory efficient and fast version of SPECK coder is proposed. The proposed coder uses fixed size static memory, which stores markers to facilitate coding. Replacement of data dependent lists with small fixed size static memory reduces the memory access time, thereby making it faster than the original SPECK. The proposed coder is memory efficient and requires only one bit per pixel memory (12.5% of memory required to store image) to store markers, while coding efficiency and scalability property of the SPECK algorithm is retained thereby making it suitable for resource constrained portable hand held device and wireless sensor networks.

Keywords: Image coding, wavelet, SPECK, Listless image coder, memory efficient image coder, Block coder

I. INTRODUCTION

Transmission of images over internet and cellular networks through handheld mobile/ portable multimedia devices, are growing exponentially. Also emerging wireless multimedia sensor network (WMSN's) require real time image transmission among its nodes and hubs over wireless channel. These devices and sensors are constrained in terms of memory, battery life and processing power. Transmission of images through these devices over internet and wireless channels require an embedded image compression algorithm which is efficient and requires minimal resources (memory, computational power and battery lifetime) [1-5].

State of art image coders such as JPEG2000[6] and EBCOT[7] are coders with increased computational complexity, while wavelet based coders Set partitioning in hierarchal trees (SPIHT)[8] and set partitioning embedded block coder (SPECK)[9] coders are most suitable coders among the wavelet based coders for resource constrained handheld devices and WMSN's[4] [10].

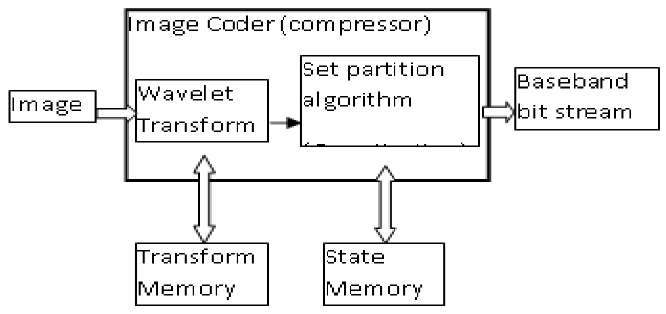


Figure 1: Block diagram of wavelet based set portioning coders

Wavelet based coders achieve compression by grouping a large number of insignificant coefficients either in form of zero-trees or zero-blocks and use data dependent list (dynamic state memory) to store state of sets and coefficients to be tested for their significance. Fig. 1 shows the block diagram of wavelet based set partitioning coders. These coders require memory for wavelet transformation and use state memory in form of various list to store state of sets and coefficients during coding. Use of continuously growing list memory in these coders, not only result in dynamic state memory requirement, but also necessitate the need for memory management. Multiple memory access, memories append and memory management contributes significantly to computational complexity of the coder. These problems become more severe for high resolution images.

Many listless versions of zero tree and zero block coders [11-14] have been reported earlier. All these coders use fixed size state tables or markers to keep track of set partitioning. However the state memory requirements of these coders are high for low memory applications. NLS [12], a listless implementation of SPIHT uses 4 bit per coefficient marker memory (50 % of the memory to store image), while implementation in [13] uses 3 bit per coefficient state memory (37.5 % of the memory to store image). LSK [14] a listless version of SPECK algorithm uses 2 bit per coefficient state memory (25 % of the memory to store image) to facilitate encoding.

In this paper, we propose a listless implementation of SPECK algorithm for efficient coding of wavelet transformed images that uses one bit per pixel marker (12.5 % of the memory to store image) to keep track of blocks and coefficients to be tested for their significance. The proposed algorithm is termed as **M**odified **L**istless **S**PEC**K** coder (MLSK) due to the obvious reasons.

The paper is organized as follows. Section II presents the overview of SPECK and LSK algorithm. Section III describes the proposed MLSK algorithm including the linear indexing property of wavelet coefficients. Simulation results and related discussions are presented in section IV and finally the paper is concluded in section V.

II. OVERVIEW OF SPECK AND LSK ALGORITHM

SPECK [9] is bit plane coding algorithm and encodes significance map of bit planes in decreasing order of their importance as shown in Fig. 2(a). SPECK coder uses two types of set: *S* and *I* as shown in Fig. 2(b). In the process of coding, *S* sets are partitioned by quad partitioning scheme while *I* sets are partitioned by octave band partitioning scheme as shown in Fig. 2(c) and Fig. 2(d). Each pass of SPECK comprises of sorting, and refinement. It uses two lists: list of insignificant sets (LIS) and list of significant pixels (LSP) to store state of sets and pixels. The coding algorithm proceeds as follows.

The algorithm initializes with defining initial threshold, which depends on the bit plane. The LIS is initialized with a LL-sub band. In the sorting pass, sets of LIS are tested against threshold and their significance is encoded. A significant *S* set is partitioned into four equal sets and parent set is removed from the LIS. Each of the newly formed set is tested for their significance and insignificant is added to LIS while significant set is recursively partitioned till a coefficient is reached. For a significant coefficient, sign bit of the coefficient is also coded and coefficient is send to LSP. After all sets of LIS are tested for the significance, set *I* is tested for significance. A significant *I* set is partitioned the octave band partitioning resulting in three sets and a reduced *I* set. The newly formed sets are processed in regular image scanning order.

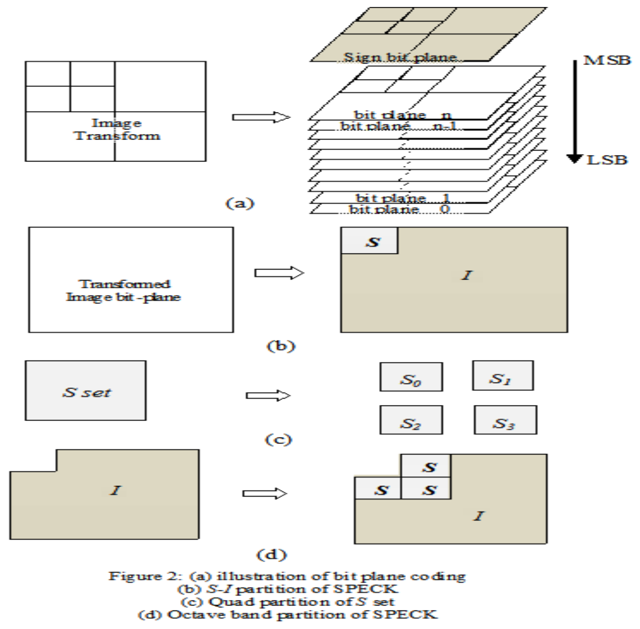

Sign bit plane
MSB
LSB
Image Transform
bit plane n
bit plane n-1
bit plane 1
bit plane 0
(a)
Transformed Image bit -plane
S
I
(b)
S set
S0
S1
S2
S3
(c)
I
S
S
S
I
(d)
Figure 2: (a) illustration of bit plane coding
(b) S-I partition of SPECK
(c) Quad partition of S set
(d) Octave band partition of SPECK

Figure 2: (a) illustration of bit plane coding
(b) *S-I* partition of SPECK
(c) Quad partition of *S* set
(d) Octave band partition of SPECK

After all sets of LIS are processed, the refinement pass is initiated which refines the quantization of the coefficients in the LSP found significant in earlier passes. The threshold is then reduced by a factor of two and the sequence of sorting and refinement passes is repeated. Sets of LIS are processed in increasing order of their size. The whole process is repeated until the desired bit rate is achieved.

Though SPECK is an efficient algorithm, large dynamic state memory requirement and increased computational complexity limits application of SPECK coder in memory constrained environment such as handheld multimedia devices and WMSN's.

LSK [14] is listless version of SPECK algorithm but does not use *I* partitioning of SPECK. State information of coefficient /block is kept in a fixed size array, with two bits per pixel of image to enable fast scanning of the bit planes. In LSK, efficient skipping of blocks of insignificant coefficients is accomplished using linear indexing [15]. The following markers are placed in the 2 bit per coefficient state table mark, to keep track of the set partitions.

MIP : The pixel is in-significant or untested for this bit-plane.

MNP: The pixel is newly significant & will not be refined for this bit-plane.

MSP : The pixel is significant and will be refined in this bit -plane.

MS2: The block of size 2×2, i.e., 4 elements to be skipped.

MS2 markers are successively used to skip 4×4 block, 8×8 block, and so on.

LSK coder uses fixed size array of two bits per coefficient to store markers facilitating coding. For a 512×512 image decomposed to 5 levels, the LSK requires 64 KB memory to store markers while for image size of 1024 x 1024, it is 256 KB which is significantly higher for memory constrained handheld devices. Also LSK does not use *I* sets of SPECK thereby generating more bits in earlier passes.

III. PROPOSED MLSK ALGORITHM

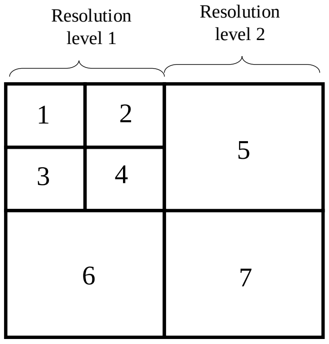
The proposed MLSK algorithm is a novel listless implementation of SPECK algorithm with small fixed size state memory. It uses dyadic wavelet transform with lifting scheme, which is a fast implementation of conventional wavelet transform and reduces memory requirement of the transform. The transformed image is then converted into linear index [12] [15]. The algorithm achieves functionalities of LIS by using fixed size marker array and removes LSP by merging refinement pass in sorting pass. The concept of linear indexing and detailed algorithm are discussed below.

Linear Indexing:

The linear indexing allows addressing a dyadic transformed wavelet coefficient by a single index instead of two. Let NxN be the size of the transformed image $\{\Im_{i,j}\}$, and let r and c be the row and column indices of a particular wavelet coefficient. The linear index i of transformed image $\{\Im i\}$, varying in the range of 0 to N^2-1, can be obtained by simply interleaving the bits of binary representation of r and c [12]. Fig. 3(a) shows linear indexing of an 8x8 image with two level dyadic transform in Z scan order. Fig. 3(b) shows sub bands at different resolution levels of the image. The linear indexing uses Z scan order to enlist coefficients of sub bands consecutively in their wavelet coefficients pyramidal structure.

	0	1	2	3	4	5	6	7
0	0	1	4	5	16	17	20	21
1	2	3	6	7	18	19	22	23
2	8	9	12	13	24	25	28	29
3	10	11	14	15	26	27	30	31
4	32	33	36	37	48	49	52	53
5	34	35	38	39	50	51	54	55
6	40	41	44	45	56	57	60	61
7	42	43	46	47	58	59	62	63

(a)



(b)

Figure. 3: Illustration of linear indexing for an 8x8 image with 2-level DWT (a) linear index, (b) sub bands at different resolution levels

MLSK Algorithm:

Let a zero mean square image X of size (NxN) with $N=2^p$, that after L level of dyadic Lifting wavelet transform is read into the linear array $\{\Im_i\}$, using linear indexing having $N_{pix}=N^2$ coefficients. The LL- band has $N_{pix}/4^L$ coefficients. MLSK algorithm uses set structures and partitioning scheme of SPECK algorithm. The functionality of LIS is obtained by using a fixed size marker array. In MLSK, state of blocks of size 4 (2x2 block) is stored instead of coefficients, thereby reducing the memory requirement of propose coder. Use of LSP is avoided in the MLSK by merging refinement pass in the sorting pass.

The following markers are placed in the state memory 'mark' of size $N_{pix}/4$, to keep track of the set partitions. Each marker and its meaning is listed below.

MSB:	The block of size 4 (2x2 block) is found significant in early passes
MI:	Marker used at the beginning of each insignificant *I* block.
MS^q:	Markers used at the beginning of each insignificant block of size 4^q (2^qx2^q block). These markers indicate the block size of insignificant set. MS^1:block of size 4 (2x2 block), MS^2: block of size 4^2 (4x4 block) and so on

The encoder algorithm shown by flowchart in Fig. 4 is performed for each bit plane starting with initial threshold T where $T=2^n$ and decrementing up to 0 or until a bit budget is achieved. The algorithm begins by initializing the static array 'mark' of size $N_{pix}/4$, mapping all possible block of size 4 (2x2 block) of the image. Initially LL-band marker is set [mark(0)=$MS^{(p-L)}$], first *I* set marker is set as MI [mark($4^{(p-L-1)}$)= MI] and rest of the elements of the 'mark' set to '0'.

In MLSK significance of a set or block B against a threshold T= 2^n is given as,

$$\psi_n(B) = \begin{cases} 1, & if \quad T \le \max_{i\in B}(|\Im_i|) < 2T \\ 0, & if \quad \max_{i\in B}(|\Im_i|) < T \end{cases} \quad ......(1)$$

In sorting pass, all the elements of state memory 'mark' are scanned. For mark (j)=MS^q, the block, the size of the block is specified by the marker. The block is tested against threshold and its significance is encoded. An insignificant block is skipped and mark index is incremented accordingly. A significant block is partitioned into four blocks (quad partition) and each newly partitioned block is tested against threshold and its significance is encoded. For a insignificant newly partitioned block, corresponding mark element is updated by MS* marker according to size of block. For a significant block of size 4 (2x2 block), corresponding mark is set as 'MSB' and each coefficient of the block is tested against threshold and its significance is encoded. For a significant coefficient, its sign bit is encoded.

'MSB' marker indicates that the corresponding block of size 4 has been found significant in previous passes and it contains at least one coefficient which requires refinement in the pass. A coefficient found is significant in previous passes is identified by the fact that coefficients magnitude is greater than or equal to twice of threshold, For the coefficients of the block found significant in earlier passes, refinement bit is encoded, For other coefficient of the block their significance is encoded and for significant coefficient its sign bit is encoded.

'MI' marker indicates an *I* block. The corresponding *I* block is tested against threshold and its significance is encoded. For a insignificant *I* block, the pass ends while for a significant block, octave band partitioning is effected by setting corresponding mark element for three newly formed square block as corresponding MS* marker and for new *I* block as MI if the reduced block exits. Then the testing of *S* blocks proceeds as explained above.

MLSK algorithm is symmetrical and its decoder follows the same overall procedure as the encoder with some low-level changes. To decode, (use input) instead of output, and set the (bits) and signs of coefficients. The decoder performs mid-tread de-quantization for coefficients that are not fully decoded

As the MLSK use set partitioning rules of SPECK, both coder produces the exact same output bits in each pass, though in a different order because refinement is merged in sorting pass as shown in Fig. 5. Thus slight degradation of PSNR may occur in MLSK than SPECK, if bit budget is exhausted in the middle of pass as bits will also be used for refinement. However, at the end of sorting pass, MLSK encodes same information as that of SPECK. Thus the MLSK generates embedded bit stream with progressive transmission using small fixed size markers,

The use of small fixed size memory reduces the computational cost involved in multiple memory access and appending the dynamic memory of SPECK. The coding complexity of MLSK coding algorithm is at par with LSK due to less memory access in MLSK and also due to use of different MS* markers while in LSK, skipping of bigger sets is achieved by using consecutive MS2 markers [14].

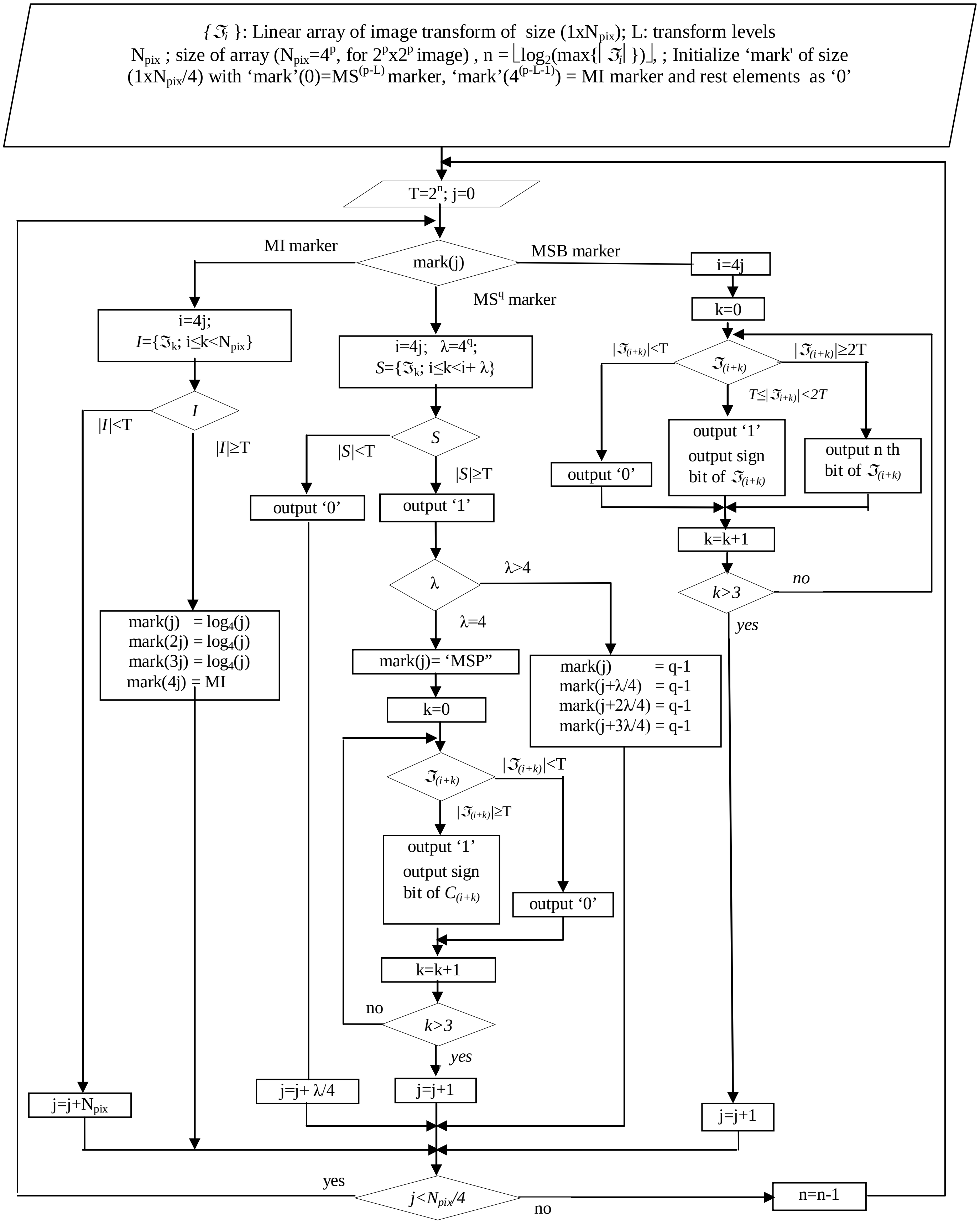

$\{\mathfrak{I}_i\}$: Linear array of image transform of size ($1xN_{pix}$); L: transform levels
N_{pix} ; size of array ($N_{pix}=4^p$, for $2^p x 2^p$ image) , $n = \lfloor \log_2(\max\{|\mathfrak{I}_i|\}) \rfloor$, ; Initialize 'mark' of size ($1xN_{pix}/4$) with 'mark'(0)=$MS^{(p-L)}$ marker, 'mark'($4^{(p-L-1)}$) = MI marker and rest elements as '0'
$T=2^n$; j=0
mark(j)
MI marker
MSB marker
MS^q marker
i=4j; $I=\{\mathfrak{I}_k;\ i \le k < N_{pix}\}$
I
$|I|<T$
$|I|\ge T$
mark(j) = $\log_4$(j)
mark(2j) = $\log_4$(j)
mark(3j) = $\log_4$(j)
mark(4j) = MI
$j=j+N_{pix}$
i=4j; $\lambda=4^q$; $S=\{\mathfrak{I}_k;\ i \le k < i+\lambda\}$
S
$|S|<T$
$|S|\ge T$
output '0'
output '1'
λ
$\lambda>4$
$\lambda=4$
mark(j)= 'MSP"
mark(j) = q-1
mark(j+λ/4) = q-1
mark(j+2λ/4) = q-1
mark(j+3λ/4) = q-1
k=0
$\mathfrak{I}_{(i+k)}$
$|\mathfrak{I}_{(i+k)}|<T$
$|\mathfrak{I}_{(i+k)}|\ge T$
output '1' output sign bit of $C_{(i+k)}$
output '0'
k=k+1
$k>3$
no
yes
j=j+ λ/4
j=j+1
i=4j
k=0
$\mathfrak{I}_{(i+k)}$
$|\mathfrak{I}_{(i+k)}|<T$
$|\mathfrak{I}_{(i+k)}|\ge 2T$
$T\le|\mathfrak{I}_{(i+k)}|<2T$
output '0'
output '1' output sign bit of $\mathfrak{I}_{(i+k)}$
output n th bit of $\mathfrak{I}_{(i+k)}$
k=k+1
$k>3$
no
yes
j=j+1
$j<N_{pix}/4$
yes
no
n=n-1

Figure 4 : flowchart of MLSK algorithm

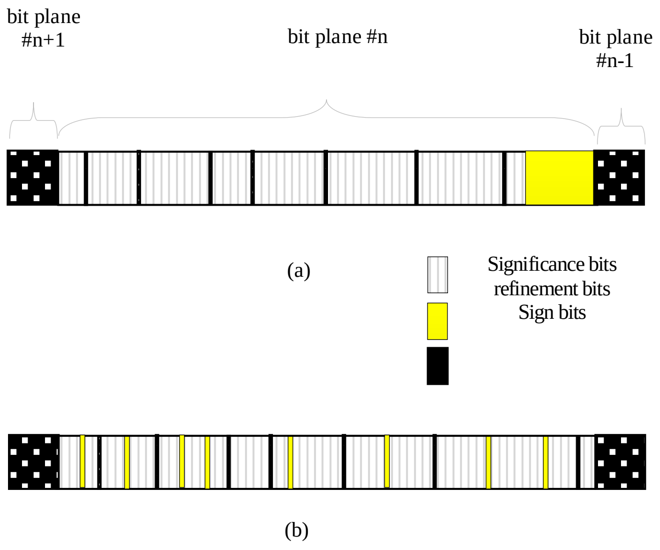


Figure 5 a bit plane in (a) SPECK (b) MLSK

State Memory:

SPECK use linked lists to store the significant information of coefficients, and blocks there by requiring a data dependent memory. The proposed MLSK coder uses a static list to store leading markers. Here the memory required for storing the state information in MLSK, LSK and SPECK are estimated. The required memory size for MLSK and LSK is fixed while for SPECK it is proportional to the number of entries in the corresponding lists (LIS and LSP).

In SPECK, each entry in LIS is the address of a square block of arbitrary size including that of a single coefficient. In actual implementation, a separate list is maintained for each block size. However, LSP contains the address of significant coefficient. Let N_{LIS} and N_{LSP} be the number of entries in LIS and LSP respectively and 'b' be number of bits required to store addressing information, then the total required memory due to lists in SPECK is given by

$$M_{SPECK} = b[N_{LIS} + N_{LSP}] \text{ bits} \qquad (2)$$

LSK uses a static memory of size equal that of coefficients array to store markers. As LSK uses two bit markers then for a image size (R x C) memory required for marker is

$$M_{LSK} = 2RC \text{ bits} \qquad (3)$$

MLSK uses a static memory of one fourth of coefficients array size, to store markers. As MLSK uses four bit markers then for a image size (RxC) memory required for marker is

$$M_{MLSK} = RC \text{ bits} \qquad (4)$$

In worst case all the coefficients may be in either LIS or in LSP in SPECK coder. For an image size of 512x512, worst case state memory required for SPECK is 576 KB, for LSK 64 KB and for proposed MLSK it is only 32 KB.

IV. SIMULATION RESULTS

The coding performance, and memory requirement, of the MLSK coder is evaluated and compared with SPECK, and LSK on three classical grayscale test images (each 512×512, 8 bits/pixel); Lena, Barbara, and Baboon. A 5-level dyadic wavelet decomposition using bi-orthogonal 4.4 filter with lifting scheme is used. Floating point, transform coefficients are quantized to the nearest integers, and read into the linear array using linear indexing. The simulations are performed using MATLAB platform on a PC with Intel CPU T 2080 @ 1.73 GHz having 512 MB RAM. All the coders are implemented on the same platform without arithmetic or context based coding. The test images are binary encoded once up to the last bit plane and are decoded at different bit rates from the same embedded bit-stream.

Coding Efficiency:

Rate-Distortion performance (coding efficiency) of MLSK, LSK and SPECK coder is measured in terms of peak signal to noise ratio (PSNR). Coding performance of image 'Lena' for various coders is given in Fig. 6. It can be observed from the figure that coding efficiency of the MLSK is at par to that SPECK. It should be noted that at the end of each bit plane coding gain obtained in MLSK and SPECK algorithms are identical and slightly better than LSK at lower bit rate. This is because MLSK coder uses the set partitioning rules of SPECK, while LSK coder does not use I partitions resulting in some more bits generated resulting in slightly lower PSNR at low bit rates. For bit rates somewhere in middle of a pass, the efficiency of MLSK is slightly lower than that of SPECK and LSK. This is because in MLSK refinement pass is merged in sorting pass and a refinement bits are spread in the bit stream. For the bit rates somewhere in the middle of pass, in the MLSK bit budget is also consumed in refinement bits thereby reducing the number of new significant bit coded. This results in reduction of PSNR as a bit representing a new significant pixel provides more improvement in PSNR than that of a bit representing a refinement pixel.

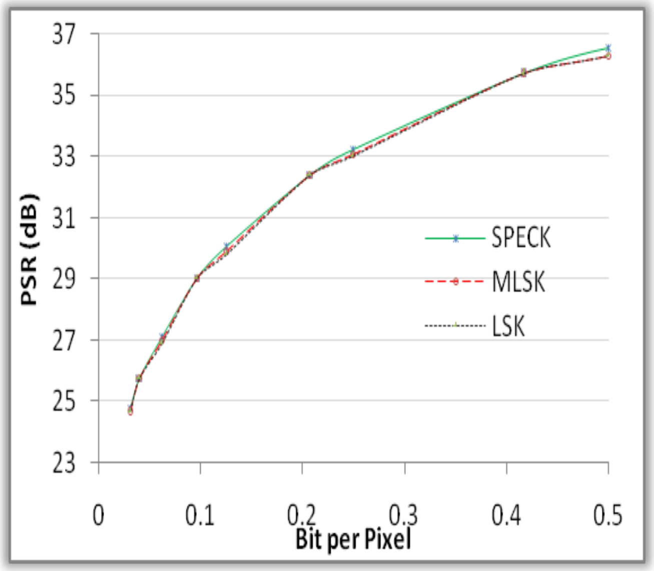


Figure 6: PSNR vs Bit rate for MLSK, SPECK and LSK for 'LENA' (512 x 512)

The coding efficiency of SPECK, LSK and MLSK coder for three test images (Lena, Barbara and Baboon) are given in Table 1. The results are without arithmetic or context based coding. It can be observed from the table that PSNR of MLSK coder is at par with SPECK and LSK. The reduction in coding gain at some points is due to bits reordering in MLSK. It should also be noted that at bit rates before the end of pass MLSK coder has slightly higher coding gain than SPECK e.g. at 0.25 bpp for image Baboon. This is because these bit budgets exhaust near the end of pass and in this condition MLSK codes most of the new significant information along with most of the refinement while in SPECK coder codes almost all new significant information are coded, but most of the refinement bits remains uncoded.

Table 1 Coding performance of SPECK, LSK, MLSK (dB)

BPP	SPECK	LSK	MLSK
LENA (512 x512)			
0.0625	27.114	26.989	26.908
0.125	30.044	29.916	29.817
0.25	33.245	33.073	33.007
0.5	36.553	36.292	36.280
1	39.709	39.535	39.484
BARBARA (512 x512)			
0.0625	22.814	22.742	22.704
0.125	24.425	24.057	24.004
0.25	27.139	26.692	26.620
0.5	30.989	30.570	30.518
1	35.885	35.438	35.344
BABOON (512 x512)			
0.0625	20.524	20.486	20.490
0.125	21.477	21.384	21.387
0.25	22.738	22.734	22.841
0.5	24.968	24.712	24.617
1	28.372	28.116	27.955

*Above results are without arithmetic / context based coding

The visual performances of MLSK coder and SPECK coder are shown in Fig, 7 for the three test images Lena, Barbara and Baboon. In subjective evaluation, the reconstructed images appear to be similar for both coders.

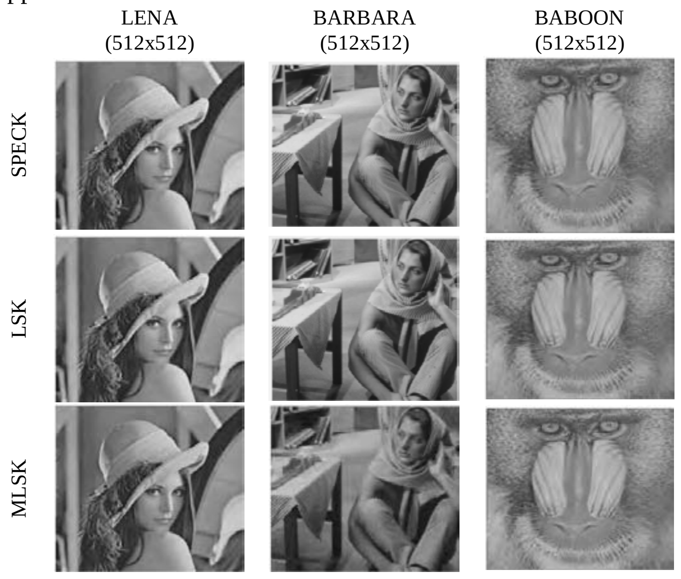


Figure. 7: comparative coding results of test images at 0.125 BPP

State Memory

The memory requirements of MLSK coder, LSK and SPECK to store state information are compared in Table 2 for the three test images. SPECK coder requires data dependent variable memory (dynamic memory) while LSK and proposed MLSK coder replace dynamic memory with fixed size static memory. From the table it is evident that in terms of processing memory MLSK coder outperforms others. LSK require 2 bit per coefficient static memory while MLSK require 1 bit per coefficient (4 bit per 4 coefficient) to store significance state.

Table 2 State Memory required (in KB)

BPP	SPECK	LSK	MLSK
LENA (512 x512)			
0.0625	13.3	64	32
0.125	24.8	64	32
0.25	46.7	64	32
0.5	88.5	64	32
1	182.7	64	32
BARBARA (512 x512)			
0.0625	16.4	64	32
0.125	35.0	64	32
0.25	53.1	64	32
0.5	102.6	64	32
1	171.8	64	32
BABOON (512 x512)			
0.0625	19.8	64	32
0.125	38.0	64	32
0.25	80.6	64	32
0.5	131.5	64	32
1	200.1	64	32

*Memory required to store image and wavelet transform is not accounted in above results

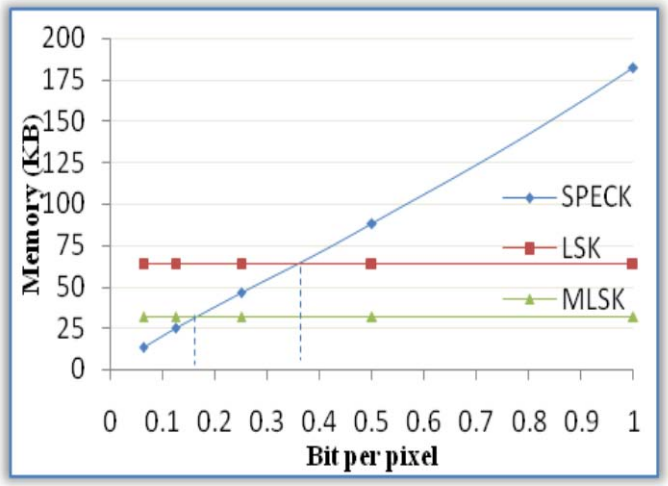


Figure 8 State Memory required (KB) for image 'LENA' (512x512) in SPECK, LSK, MLSK

Fig, 8 shows the state memory required for the coding of image Lena. It can be observed from the figure that state memory required for SPECK coder increases with bit rate while for LSK and MLSK, it is of fixed size, From the figure it is evident that in terms of processing memory, LSK outperforms SPECK for bit rates higher than 0.35 bit per pixel while proposed MLSK coder outperforms SPECK for bit rates higher than 0.15 bit per pixel. Thus MLSK coder is more suitable for low bit rate coding than LSK

V. CONCLUSIONS

In this paper we have proposed a novel implementation of listless SPECK using fixed size markers memory (one bit per pixel). It is observed that proposed MLSK coder outperforms SPECK and LSK in terms of memory requirements while coding efficiency is at par with that of SPECK. MLSK coder generates same size of bit stream as that of SPECK in a pass but in different order due to merging of refinement pass into sorting pass. Bit ordering of SPECK offers slightly better coding gain in the middle of a pass, but at the end of each bit plane coding gain is exactly identical. In terms of computational complexity proposed coder is at par with LSK while outperforms SPECK coder. Due to low memory requirement and low computational complexity, the MLSK coder is suitable for resource constrained devices such as portable camera, PDAs and wireless multimedia sensor networks..

VI. REFERENCES

[1] D Lee, H Kim, M Rahimi, D Estrin and J. D. Villasenor, “Energy efficient image compression for resource-constrained platforms”, IEEE Trans. Image Process, vol. 18, No. 9, pp. 2100-2113, Sept. 2009.

[2] I. F. Akyildiz, T. Melodia, K. R. Chowdhury, “A survey on wireless multimedia sensor networks”, Computer Networks (Elsevier) Journal, Vol. 51, No.4, pp. 921-960, March 2007.

[3] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, “Wireless Multimedia Sensor Networks: Applications and Testbeds”, Proceedings of IEEE, Vol. 96, No.10,pp. 1588-1605, October 2008.

[4] Li Wern Chew, Li-Minn Ang, and Kah Phooi Seng, ”Survey of image compression algorithms in wireless sensor networks.”: International Symposium on Information Technology Malaysia, 2008. pp. 1-9. 2008.

[5] Satyajayant Misra, Martin Reisslein, and Guoliang Xue, “A Survey of Multimedia Streaming in Wireless Sensor Networks”, IEEE Communication Surveys & Tutorials, Vol. 10,No. 4 pp. 18-39, 2008.

[6] D. Santa-Cruz, R. Grosbois and T. Ebrahimi, “JPEG 2000 performance evaluation and assessment,” Signal Processing: Image Communication, vol. 17, pp. 113–130, Jan. 2002.

[7] D. Taubman.: ‘High performance scalable image compression with EBCOT’, IEEE Trans. Image Process., 9, pp. 1158–1170, 2000.

[8] A. Said, W. A. Pearlman: ‘A new fast and efficient codec based on set partitioning in hierarchical trees’, IEEE Trans. Circuits Syst. Video Technol., vol. 6, pp. 243–250, 1996

[9] W. A. Pearlman, A. Islam, N. Nagaraj, A. Said: ‘Efficient low complexity image coding with set-partitioning embedded block coder’, IEEE Trans. Circuits Syst. Video Technol., 14, pp. 1219–1235, 2004.

[10] R.Sudhakar, Ms R. Karthiga, S.Jayaraman, “Image Compression using Coding of Wavelet Coefficients–A Survey”, ICGST-GVIP Journal, Vol. 5, No. 6 pp. 25-38, June 2005.

[11] W. K. Lin and N. Burgess, “Listless zerotree coding for color images”, In Proc. of the 32nd Asilomar Conf. on Signals, Systems and Computers, vol. 1, pp. 231-235, Nov 1998.

[12] F. W. Wheeler and W. A. Pearlman, “SPIHT image compression without lists”, IEEE conference on acoustics, speech and signal processing (ICASSP2000), vol. 4, pp. 2047-2050, May 2000.

[13] H. Pan, W. C. Siu, et al.. "A fast and low memory image coding algorithm based on lifting wavelet transform and modified SPIHT" Signal Processing: Image Communication, Vol. 23, No.3, 146-161, 2008.

[14] M. V. Latte, N. H. Ayachit and D. K. Deshpande, “Reduced memory listless SPECK image compression”, Elsevier’s Journal of Digital Signal Processing, vol. 16, issue 6, pp. 817-824, Nov. 2006.

[15] G. Seetharaman and B. Zavidovique, “Z-trees: Adaptive pyramid algorithms for segmentation,’’ in Proc. of the International Conf. on Image Processing (ICIP-98), pp. 294-298, 1998.