International Journal of Engineering Science Invention (IJESI)
ISSN (Online): 2319 – 6734, ISSN (Print): 2319 – 6726
www.ijesi.org ||Volume 7 Issue 3 Ver. II || March 2018 || PP29-36

A Parallel Approach For Maximum Quantization Of Descendants Of Wavelet Trees

E. Sudarshan[1], K. Seena Naik[2]
[1, 2] (Department of Computer Science and Engineering, S R Engineering College, Warangal, INDIA)
Corresponding Author: E. Sudarshan

Abstract : *The Lossless Image Compression Methods Have High Demand, Particularly In A Medical Imaging Systems. Therefore, We Need To Increase The Image Compression Algorithm Acceleration Given The Demand In The Future Than The Present Demand. General Purpose Graphics Processing Unit (Gpgpu) Is An Efficient With The Development Of Computing Technologies. Therefore, The Gpgpu Is Used To Image Compression Algorithms To Achieve Efficiency And Speed In Performance. The Backward Wavelet Tree Based Image Compression Algorithm Uses A Compute Unified Device Architecture (Cuda) Platform For Processing It. After Employing The Dyadic Partition, The Algorithm Constructs The Wavelet Tree And Used To Process In The Backward Manner. However, The Maximum Quantization Of Descendants (Mqd) Is Found In The Tree And It Will Process, With The Only Three Levels Of Mqd's And The Corresponding Image Coefficients. The Parallel Mqd Algorithm Achieves Acceleration 2.83 Times Faster Than The Serial Mqd And Moreover, As The Image Size Increases, The Algorithm's Speed Improves.*

Keywords-*Wavelet Trees, Maximum Quantization Descendants, Parallel Architecture, Cuda, Gpgpu.*

Date of Submission: 20-02-2018 Date of acceptance: 07-03-2018

I. Introduction

In recent years, computationally compression is still needed in medical imaging applications, notwithstanding the development of computing performance and storage costs [1]. One of the significant applications is Electronic Healthcare Record (EHR) maintenance is the biggest issue since each citizen full-length record may have huge. Each EHR has a lot of radiological images, and they have more space. To avoid cost-effective image compression model on all factors. Discrete Wavelet Transform (DWT) is widely used in a lossless image compression [2], but algorithms needed more compression and speed of the algorithm's performance. The DWT compression is usually two steps: (1) transition phase and (2) an encoding platform. In the transition phase, the reversible 5/3 lifting method is employed for compression [3] without loss of wavelets. The 5/3 wavelet integer for integer transition can be converted to integer coefficients which cannot be lost to the integer pixel value [4]. The 5/3 wavelet training scheme uses at least memory and requires less computation than the traditional transformation system. The proposed zero-tree based algorithms are called SPIHT [5], EBCOT [6] and BCWT [7]. BCWT algorithm is an efficient and fast, this uses the maximum size of the successors of the single pass coding system. The sequential BCWT algorithm implemented in the CPU, the same BCWT implemented on the parallel platform to make a change, few stages have to reconstruct.

Currently, the General Purpose Graphical Processing Unit (GPGPU) has traditionally been used to speed up the procedures [8]. GPGPU computational functions now make compression algorithms significantly faster than the serial algorithms, such as JPEG2000 [9] and JPEG-XR [10]. However, these standards do not support the parallelization, since there is no internal parallel in their structure. Encoding is the most favourable MQD for parallel compression due to its inner parallelism and simplicity [7, 11]. The BCWT algorithm creates an MQD map in recursive back coding in a pass, which removes all wavelength trees before coding [5, 6]. See the complete explanation of BCWT in [7].

The parallel-based MQD is proposed with a one-pass encoding system (i.e., all bands have a corresponding MQD matrix) and employed in an encoding or a decoding system. The GPGPU model CUDA [12] is described in the second section with an internal structure. The third section defines the internal structures of the sequential BCWT. To accelerate the sequential algorithm, the MQD stage has been designed and changed as parallel, this is explained in section four. The fifth section discusses the result analysis and the future work and conclusion has made in the sixth section.

II. Cuda Architecture

Due to its diversity, GPGPU is the best construction to speed up parallel algorithms; All devices can easily handle high-resolution images or have better performance in the video. GPGPU is traditionally designed for graphical tasks, but with the help of a different structured platforms, general purpose tasks can be done. Parallel Computing Platform Frames are (1) NVIDIA Compute Unified Device Architecture (CUDA) and (2) Kharos Group's Open Computing Language (OpenCL) [13]. Finally, CUDA has chosen to develop this work, since online classes, blogs, tutorials, and forums are widely available. The following Fig.1 shows the interior structure of the CUDA including their respective memories, threads, blocks, and grids.

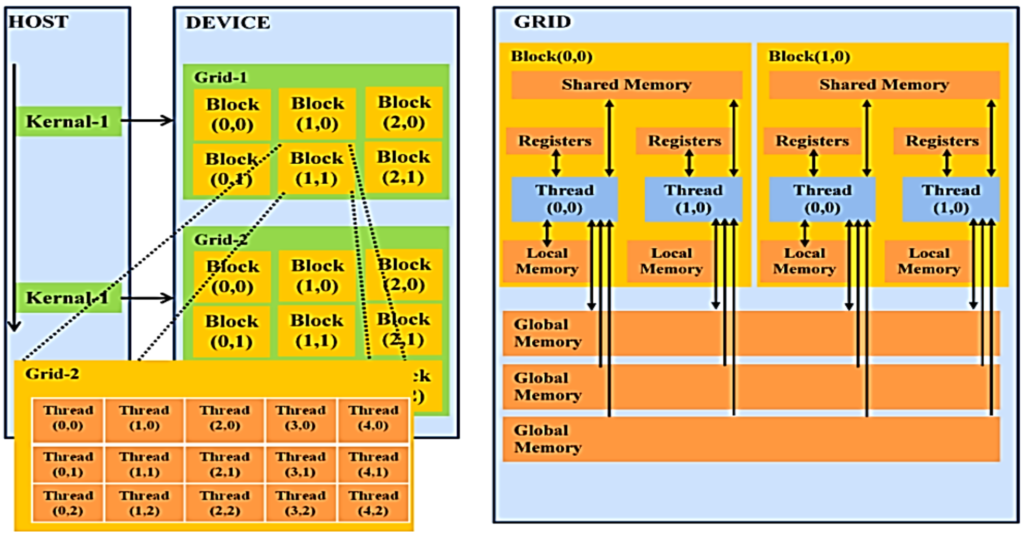


Figure 1 CUDA Hierarchical Structure

The General Purpose Graphical Processing Unit (GPGPU) promotes graphics-based tasks, but also maintains general purpose tasks. The GPU is a better computational resource processor. A full-length of parallel processors was developed by adding the fixed and a special purpose features of the processor. Initially, we started organizing the graphics related applications through a more flexible pipeline structures. In the end, it became a strong architecture, that is, the GPU. Basically, GPGPU has two programming models: (1) Compute Unified Device Architecture (CUDA) developed by NVIDIA Corporation and (2) OpenCL developed by the Khronos Group. In both, we selected CUDA to implement the parallel steps of this theory.

There are many types of memories in the general structure of the CUDA. Different types of memory (1) Global memory, (2) shared memory, (3) fixed memory, (4) local memory. Here, the CPU maintained memory is called "host memory" and the referred GPU memory is called "device memory", that collects the data from host memory. before performing the task calculations. Generally, among the memories, the global memory is the largest memory and also its an expensive operation, complexity but it gives more benefits by accessing the data from global memory as large as possible. If any variable is trying to accessing the same location or data, it may be a problem, so that the global memory must be combined.

III. Sequential Backward Coding Of Wavelet Trees

In 2005, Texas Tech University found the basic BCWT algorithm [7] and consistently added the best features to the codec. The best feature is indicated for QP or Qmin parameter bit-rate control. When Qmin is set to zero, an image quality is as usual (lossless) otherwise the image quality is lossy. BCWT algorithm the usual block diagram is shown in Fig.2. The wavelet tree-based coding algorithms are EZW, SPIHT etc., and their variations are begin the process of encoding from the top to the bottom in the wavelet tree. Therefore, the wavelet tree should scan the complete the repeated coding process. The algorithm uses more resources by checking the tree several times, such as time, energy, memory, counts, etc. The BCWT algorithm exceeds those problems.

The BCWT algorithm is a completely reversible mechanism that enables the encoding process from the bottom level of wavelet tree to the top and that to it will scan only one once. The entire encoding process in BCWT is done in the same backward pass, and thus, it is very efficient.

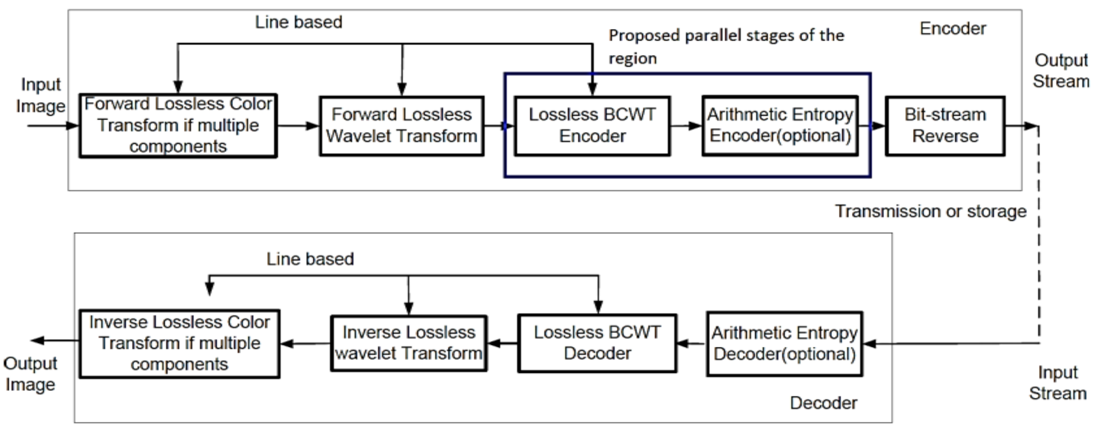

Line based
Proposed parallel stages of the region
Encoder
Input Image
Forward Lossless Color Transform if multiple components
Forward Lossless Wavelet Transform
Lossless BCWT Encoder
Arithmetic Entropy Encoder(optional)
Bit-stream Reverse
Output Stream
Transmission or storage
Line based
Inverse Lossless Color Transform if multiple components
Inverse Lossless wavelet Transform
Lossless BCWT Decoder
Arithmetic Entropy Decoder(optional)
Output Image
Input Stream
Decoder

Figure 2 Lossless BCWT block diagram

The BCWT algorithm uses high-frequency coefficients after the partition of an image. Based on these modules the BCWT designed by using the MQD and MQL algorithms.

The BCWT algorithm [11] terminology:

$C_{i,j}$: (i, j) coordinate of a wavelet coefficient.

$O_{i,j}$: (i, j) all the offspring of a coefficient set.

$D_{i,j}$: (i, j) all the descendants of a coefficient set.

$L_{(i,\,j)} = D(i, j) - O(i, j)$: (i, j) all the leaves of a coordinate set.

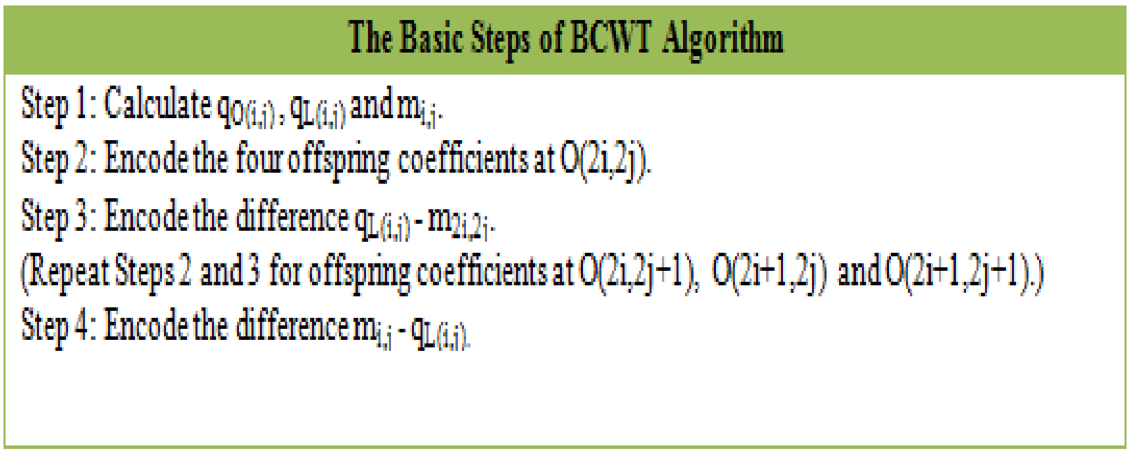
$q_{i,j} = \begin{cases} \lfloor \log_2 |c_{i,j}| \rfloor, if\,|c_{i,j}| \geq 1 \\ -1 \qquad\quad , otherwise \end{cases}$: The quantization level of the coefficient $c_{i,j}$.

$q_{O(i,j)} = \max_{(k,l)\in O(i,j)}\{q_{k,l}\}$: The maximum quantization level of the offspring of (i, j).

$q_{L(i,j)} = \max_{(k,l)\in L(i,j)}\{q_{k,l}\}$: The maximum quantization level of the leaves of (i, j).

q_{min}: The minimum quantization threshold.

$m_{i,j} = \begin{cases} q_{O(i,j)}, if\,(i,j) isin\ theleveltosubbands \\ \quad \max\{q_{O(i,j)}, q_{L(i,j)}\},\ otherwise \end{cases}$:The node in MQD $m_{i,j}$.


The Basic Steps of BCWT Algorithm

Step 1: Calculate $q_{O(i,j)}$, $q_{L(i,j)}$ and $m_{i,j}$.

Step 2: Encode the four offspring coefficients at O(2i,2j).

Step 3: Encode the difference $q_{L(i,j)}$ - $m_{2i,2j}$.

(Repeat Steps 2 and 3 for offspring coefficients at O(2i,2j+1), O(2i+1,2j) and O(2i+1,2j+1).)

Step 4: Encode the difference $m_{i,j}$ - $q_{L(i,j)}$.

BCWT coding units have many smaller branches of the wavelet tree of their respective MQD map nodes and have repeatedly been processed. Fig.3 shows the coding unit of U (i, j) in the HL subband. It will display intermediate coefficient coordinates.

Table I list of steps is used to encode the BCWT coding unit U (i, j). Level 3 subband (i, j), additional steps are taken to calculate specific MQD nodes. Since each coding unit has four MQD nodes derived with its children, they are not low units and three units for children. Once a level 3 unit is coded, it is necessary to know that these four MQD nodes are not required, and the MQD will not be placed on the map, is discussed in section 4. As a result, the BCWT's MQD map is only 1/15 of size with size, and 1/4 is not usually the size [7].

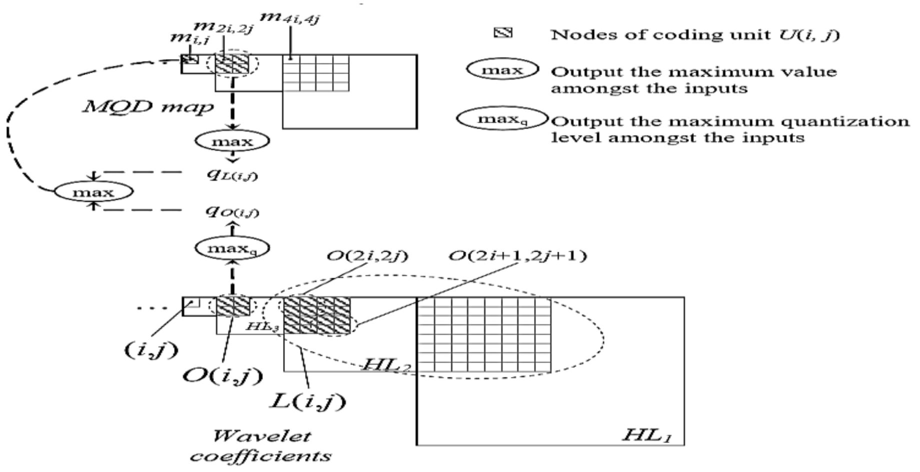
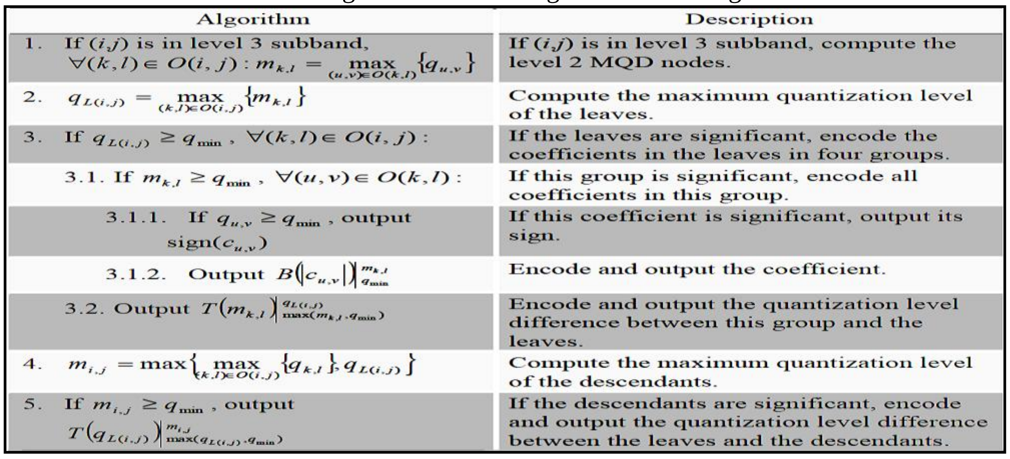


Figure 3 BCWT coding unit procedures

Table I. Algorithm of encoding a BCWT coding unit



Algorithm	Description
1. If (i,j) is in level 3 subband, $\forall(k,l)\in O(i,j) : m_{k,l} = \max_{(u,v)\in O(k,l)}\{q_{u,v}\}$	If (i,j) is in level 3 subband, compute the level 2 MQD nodes.
2. $q_{L(i,j)} = \max_{(k,l)\in O(i,j)}\{m_{k,l}\}$	Compute the maximum quantization level of the leaves.
3. If $q_{L(i,j)} \ge q_{\min}$, $\forall(k,l)\in O(i,j)$:	If the leaves are significant, encode the coefficients in the leaves in four groups.
3.1. If $m_{k,l} \ge q_{\min}$, $\forall(u,v)\in O(k,l)$:	If this group is significant, encode all coefficients in this group.
3.1.1. If $q_{u,v} \ge q_{\min}$, output sign$(c_{u,v})$	If this coefficient is significant, output its sign.
3.1.2. Output $B(\lvert c_{u,v}\rvert)\Big\vert_{q_{\min}}^{m_{k,l}}$	Encode and output the coefficient.
3.2. Output $T(m_{k,l})\Big\vert_{\max(m_{k,l},q_{\min})}^{q_{L(i,j)}}$	Encode and output the quantization level difference between this group and the leaves.
4. $m_{i,j} = \max\{\max_{(k,l)\in O(i,j)}\{q_{k,l}\}, q_{L(i,j)}\}$	Compute the maximum quantization level of the descendants.
5. If $m_{i,j} \ge q_{\min}$, output $T(q_{L(i,j)})\Big\vert_{\max(q_{L(i,j)},q_{\min})}^{m_{i,j}}$	If the descendants are significant, encode and output the quantization level difference between the leaves and the descendants.

IV. Mqd Approaches

Serial Approach: After the dyadic decomposition, the significant coefficients are take into the consideration to encodes the BCWT process. In each level, high-frequency coefficients are sub-banded at first. This BCWT encoding algorithm [11] unit is discussed in the Table I. In the BCWT algorithm, the Maximum Quantization of Descendants approach is used to encode every four coefficients with the corresponding MQDs independently as like SPIHT algorithm, the backward coding scheme in BCWT. This procedure generates two types of bit-streams, (1) The coefficient bit-stream, and (2) MQD bit-stream. The first type of bit-stream shows the clause 3.1 to encode the coefficient bits and the sign bit, as like a LIP and a LSP process of the SPIHT algorithm. The second type of bit-stream shows the clauses 3.2 and 5 to encode the coefficient's quantization level differences, as like a SPIHT LIS process of type-B and LIS process of type-A. The clause 3.2 encodes the differences in quantization levels of a qL(i, j) and max (mk,l, qmin), as like SPIHT LIS process of type-B node. In the BCWT procedure clause 5 finds the quantization levels and finds the differences between mi, j, and max (qL (i, l), qmin), as like a SPIHT LIS process for type A.

For an instance, as per the clause 5, if a set is not a significant in the last two continuous bit-plane and then it will make a significant, codeword '001' of the MQD bit-stream. The bit-stream of SPIHT would be distributed in various locations, which are coded; e.g. two '0's and '1'. The BCWT obtains the absolute coefficient levels from the coding information of MQD bit-stream. These coefficients are correctly decoded after the allocated bit by the expected quantization levels. There is a redundancy in the SPIHT algorithm as per [14] discussion. If a root and all its sub-tree descendants are '0's and then it's set a '0' in the encoded bit-stream. But the BCWT algorithm does not use a bit - plane coding because of an excess of redundancy. The 'zero tree detection' technique has exploited the redundancy, where would be in the tree and it is encoded one '0' for all zero trees.

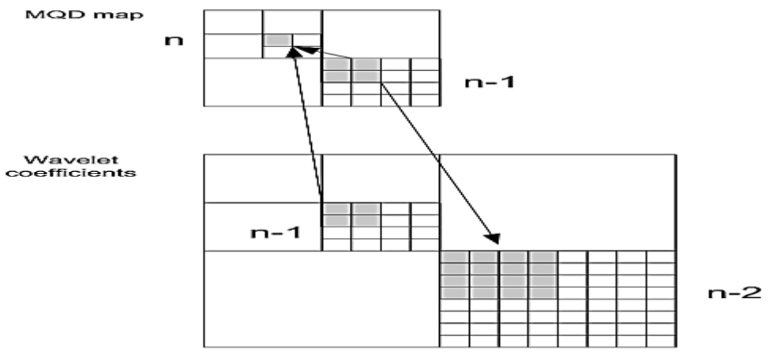


Figure 4 An instance of a HH sub-band coefficients

For instance, an HH sub-band in Fig: 4. The coding unit is processing with 25 values (which are shown in gray boxes), 20 wavelet coefficients and with 5 MQD map nodes. At (n-1)th level, 16 wavelet coefficients of n-2 level used to encode for every four MQD nodes of n-1 level. At nth level, one MQD node created using 4 wavelet coefficients of n-1 level. In the background scheme, once coding finished for all units on one level, further the wavelet coefficients are no more required so all can be cleared from memory. Ultimately, the same memory reused for further process, unlike the traditional algorithm SPIHT.

Therefore, the BCWT coding method is the memory saver and also this can start the coding at level three, but forward coding should wait for up to the store all wavelet coefficients in the memory and it has done with less latency.

Parallel Approach:
The same algorithm has been remodelled into a parallel algorithm to speed up performance. This algorithm the most simplified procedure to avoid the difficulties and it uses the top levels of bands such as HL, LH, HH, and LL coefficients along with their corresponding MQD bands. The MQD matrix formed from the low to the high level.

It has two types MQD bands: with and without corresponding lower level MQD values. In the MQD calculation, only coefficients are involved in without MQD band such as only 4 coefficients used. In the with MQD band, which uses both coefficients and their corresponding MQD values, means that, the every coefficient were a set of MQD values as per the level. With and without low-level MQD band element calculation portion has shown in Fig.5 with Lv2 and Lv1 respectively.

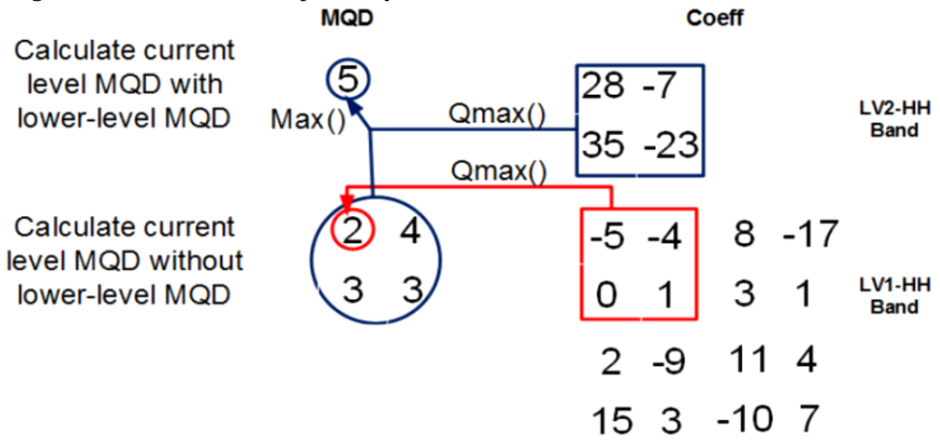


Figure 5 Calculation of two types of MQD bands

The Qmax () function is employed to determine the maximum Quantization Level (QL) of a present level of four-coefficient unit in a parallel manner [15]. The original coefficients will be stored with the minimum number of bits. However, one bit is enough for minimum value to represent a coefficient and for 0's -1 a defined in QL, as shown in Fig.5. The function Max() is used to find the maximum coefficient value among the group of numbers [16].

To calculate without MQD band values, for instance, four coefficients (-5, -4, 0, 1)'s of MQD and QL coefficient calculation: The maximum QL out of those four QLs is 2, which means the MQD of those four coefficients is 2.

$QL(-5) = \lfloor \text{Log}_2(|-5|) \rfloor = 2,$

$QL(-4) = \lfloor \text{Log}_2(|-4|) \rfloor = 2,$

$QL(0) \overset{\text{def}}{=} -1,$

$QL(1) = \lfloor \text{Log}_2(|1|) \rfloor = 0.$

To calculate type (2) MQD, for example, the MQD of four coefficients (28, -7, 35, -23), QL of each coefficient is calculated first:

$MQL(28) = \lfloor \text{Log}_2(|28|) \rfloor = 5,$

$MQL(-7) = \lfloor \text{Log}_2(|-7|) \rfloor = 2,$

$MQL(35) = \lfloor \text{Log}_2(|35|) \rfloor = 5,$

$MQL(-23) = \lfloor \text{Log}_2(|-23|) \rfloor = 4,$

At this level, the MQD value 5 is obtained from the four MQD values (2, 4, 3, 3) as shown in Fig. 5.

Procedure in CUDA:

Every MQD band is fragmented into multiple blocks. But these bands are not having parallelism, so the CUDA able to provide the parallelism between bands by using the kernels, when an offset is set to the kernels. The sequence of the process bands are LL, HL, LH and HH, from low level to high level in the tree, as shown in Fig.4.

Five steps of parallel MQD method:

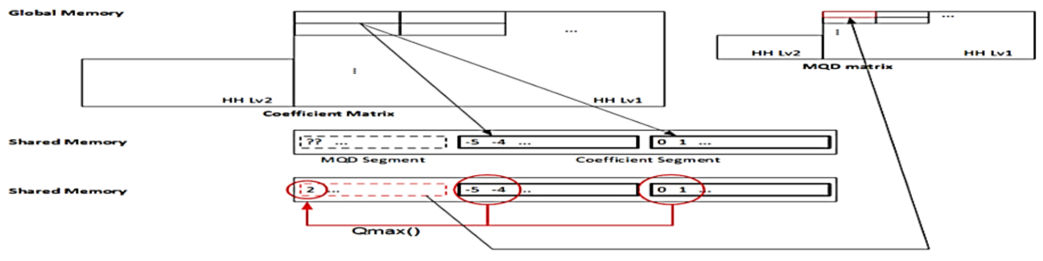
Step 1: repeat the process when in an execution; consider the bands of wavelet tree levels and their corresponding MQD levels are used to assess as per the hardware availability with respect to the memory and processor's capacity dynamically.

Step 2: initially every MQD band should fragment into multiple blocks. If the MQD band size RxC, then the Rx[C/N] blocks generated per block.

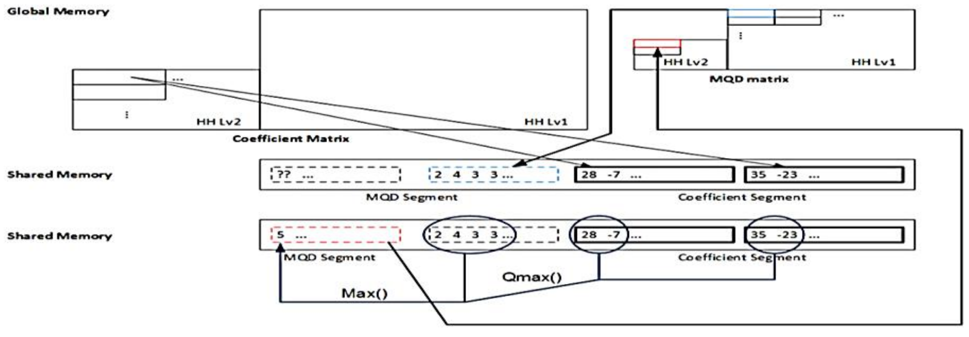
Step 3:without MQD elements the band's coefficients executes from the global memory. With MQD elements the band's coefficients executes from the global memory.

Step 4: The allocated thread produces the MQD, this has been uses the four coefficients with or without support of MQDs, which finds the maximum coefficient for calculating the QL. If the band is with MQD, then the MQD matrix will be considered otherwise which will not be considered. The maximum value will find from them.

Step 5: One block generated, once all MQDs calculated and after that to process the rest, it will back to global memory for MQD matrix. This will be depended on the offset and the size.



(a)



(b)

Figure 6 An Instance of Parallel MQD

V. Experimental Results Analysis

A parallel MQD algorithm is to achieve a faster compression when performing parallelly, which connects an acceptable running time. The parallel MQD algorithm is compared to a CPU implemented algorithm. The tremendous results were observed over the CPU based results, as shown in Fig.8.

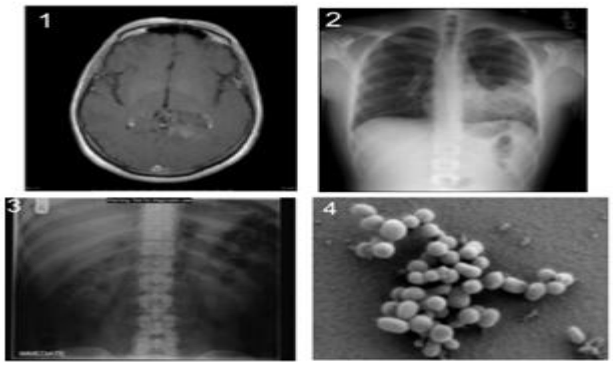


Figure 7 a four sample images

In this experiment, the four publicly available medical images used such as Brain, Chest, Spinal and Organism, as shown in Fig.7. In Table II listed the results of the test. Four pictures shown in Fig.7.1 to 7.3 are medical images, and Fig.7.4 is an organism image. Table II shows that, the PMQD algorithm's performance increases, when an image size increased. PMQD is achieved 2.8 times faster compression speed than serial MQD. If the image size is more significant than 10MB, the results will improve, as shown in Fig.8.

Table II Speed analysis between Serial and Parallel MQD

Medical Images	Memory Size (MB)	Parallel MQD	Serial MQD
Brain 2133 x 2133	4.3	15.86	26.68
Chest 2876 x 2894	7.9	28.23	64.27
Spinal 4280 x 3520	14.3	36.16	149.46
Organism 6144 x 4990	29.2	74.24	196.75

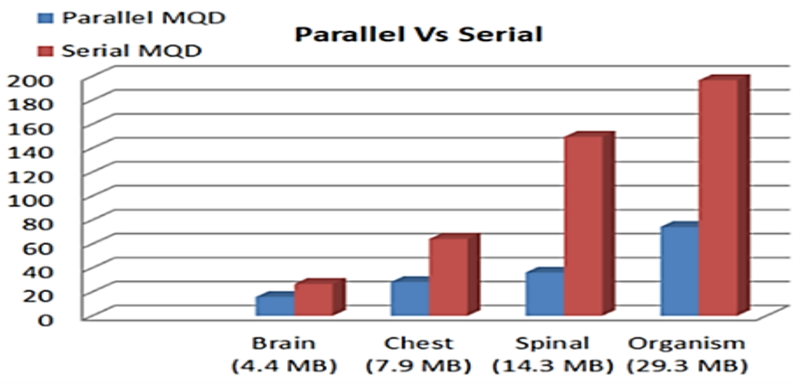


Figure 8 Comparative analysis of parallel and serial MQD

I. Conclusion

he Parallel MQD algorithm works well with a remarkable speed. If we supposed to design at the instruction level of the algorithm in parallel, we could expect the tremendous results on an MQD based coefficient encoding algorithms. As per the experimental result, the Parallel MQD achieved up 2.83 times faster coding speed reached over the serial MQD algorithm.

The additional amendments to the algorithm made by including the parallel Qmax search stage and another group-wise encoding stage [16]. Thus improves the algorithm's performance and also compression by modifying the internal bit configuration.

References

[1] D. A. Clunie, "Lossless compression of grayscale medical images", *Proc SPIE*, 2000, pp 74-84.

[2] D. F. Schomer, A A Elekes, J D.Hazeet al, "Introduction to wavelet-based compression of medical images," *Radio Graphy, no. 18*, 1998, pp. 469-481.

[3] M. D. Adams and R Ward, "Wavelet transforms in the JPEG2000 Standard," in the *IEEE PacicRim Conference on Communications, Computers and Signal Processing*, 2001.

[4] W. J. Van der Laan, A C. Jalba and J B. T. M. Roerdink, "Accelerating Wavelet Lifting on Graphics Hardware Using CUDA," *IEEE Transactions on Parallel and Distributed Systems,* Vol 22, no.l, 2011, pp. 132-146.

[5] A Said and W. A Pearlman,"A new,fast and efficient image codec, based on set partitioning in hierarchical trees", *IEEE Transactions on circuits and systems for video technology*, Vol 6, no.3, 1996, pp. 243-250.

[6] D. STaubman, "High performance, scalable image compression with EBCOT' *IEEE Trans. Image Proc., 9 (7)*, July 2000, pp 1158-1170.

[7] J Guo,S Mitra,B. Nutter ad T. Karp, "A Fast and Low Complexity Image Codec based on Backward Coding of Wavelet Trees," *Proc. Data Compression Conference (DCC) , Snowbird,Utah*,2006.

[8] NVIDIA, "Popular GPU Accelerated Application,": *http://www.nvidia.com/docs/TO/I23576/nv-aplications-cataloglowres.pdf*

[9] R Le, J.L Mundy and R L Bahar, "High Performance JPEG2K Decoder," in *IEEE 23rd International Conference on Application Specific Systems, Architectures and Processors*,2012.

[10] M. Che and J. Liang, "GPU Implementation of JPEG XR*," Proc. Of SPIE-IS&T Electronic Imaging, SPIE Vol 7543*,2010, pp.754-309.

[11] J Guo, S. Mitra, B. Nutter, T. Karp, "An Efficient Image Codec based on Backward Coding of Wavelet Trees," *IEEE Southwest Symposium on Image Analysis and interpretation*, 2006, pp.233-237.

[12] NVIDIA, "CUDA parallel computing platform ", *http://www.nvidia.com/object/cuda_home_new.html.*

[13] Khrnous group, "The open standard for parallel programming of heterogeneous systems",*http://www.khronos.orglopencl/.*

[14] A. Said and W. A. Pearlman, "A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees," *IEEE Trans. on CSVT*, vol. 6 (3), Jun. 1996, pp. 243-250.

[15] Sudarshan. E, Ch. Satyanarayana and C. Shoba Bindu, “Parallel Scheme for Multi-Pass Qmax and Multi-Rate Qmin on GPU,” *International Research, Journal of Management Science & Technology*,Vol.8 (9), Sept-2017. ISSN:2250 – 1959, 2017.

[16] Sudarshan. E, Ch. Satyanarayana and C. Shoba Bindu, “A Parallel Unit Encoding Stage for BCWT using GPGPU,” *Journal of Image Processing & Pattern Recognition Progress*, Vol.4 (3), ISSN:2394-1995, 2017.

International Journal of Engineering Science Invention (IJESI) is UGC approved Journal with Sl. No. 3822, Journal no. 43302.

E. Sudarshan "A Parallel Approach For Maximum Quantization Of Descendants Of Wavelet Trees " International Journal of Engineering Science Invention (IJESI), vol. 07, no. 03, 2018, pp 29-36