VOL. 15, NO. 9, MAY 2020 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences
©2006-2020 Asian Research Publishing Network (ARPN). All rights reserved.
www.arpnjournals.com

PARALLEL APPROACH FOR BACKWARD CODING OF WAVELET TREES WITH CUDA

Sudarshan E.[1], K. Seena Naik[2] and P. Pavan Kumar[3]
[1]Department of Computer Science and Engineering, Sumathi Reddy Institute of Technology for Women, Warangal, India
[2]Department of Computer Science and Engineering, S R Engineering College, Warangal, India
[3]Faculty of Science and Technology, IFHE Hyderabad, India
E-Mail: medasare@gmail.com

ABSTRACT

The lossless image compression methods have high demand, particularly in the medical imaging applications. Therefore, we need to increase the image compression acceleration given the demand in the future than the present demand. General Purpose Graphics Processing Unit (GPGPU) is efficient with the development of computing technologies. Therefore, the GPGPU is useful for lossless image compression algorithms to achieve efficiency and speed in performance. This paper is presented with a wavelet tree-based image compression algorithm based on Compute Unified Device Architecture (CUDA) platform. The encoding phases of the compression algorithm have been changed to parallelism and efficiency. With comparable compression ratios, our algorithm works faster than the sequential lossless JPEG-XR and BCWT algorithms. Currently, the parallel BCWT algorithm is subject to further improvement of speed and flexibility.

Keywords: parallelism, lossless image compression, CUDA, lossless JPEG-XR, GPGPU, BCWT.

INTRODUCTION

In recent days, beneficial compression is still required in medical imaging applications, despite the improvement of computing efficiency and storage costs (DAC *et al*., 2000). One of the significant applications is Electronic Healthcare Record (EHR) maintenance is the biggest issue since each patient's or citizens full-length record may have huge. Each EHR might have many radiological images, and they can take more memory space. To avoid that the cost-effective image compression model is needed to the tradeoff in all factors (Suda.e*et al*., 2017).

Discrete Wavelet Transform (DWT) is broadly used in lossless medical image compression (DFS *et al*., 1998), but algorithms require more compression and speed in performance. DWT based compression usually is two stages: (1) transition stage and (2) encoding stage. During the transition stage, reversible 5/3 lifting scheme is used for compression (MDAdams*et al*., 2001) without loss of wavelets.5/3 wavelet integer for integer transform, an integer pixel value can be transformed to lossless integer coefficients. The 5/3 wavelet lifting scheme uses the least memory and requires less computation than the conventional transformation system (WJVan*et al*.,2011).Many of the proposed wavelet zero-tree based algorithms are named SPIHT (A Said *et al*., 1996), EBCOT (DST *et al*., 2000) and backward coding wavelet treat (BCWT) (JGuo*et al.,* 2006). BCWT is an efficient and fast algorithm that uses the maximum size of the descendants of the single pass coding system. The sequential BCWT algorithm implemented on the CPU as a sequential approach, which can improve the sequential BCWT as parallel, so few steps have remodeled as parallel.

At present, the General-Purpose Graphical Processing Unit (GPGPU) computing has traditionally been used to speed up systematic procedures (NVIDIA GPU). The GPGPU computational tasks performed significantly to accelerate the existing lossless compression algorithms into parallels, such as JPEG2000 (RLE *et al*., 2012) and JPEG-XR (MChe*et al.*, 2010). However, some of these standards are not designed for efficient parallelization, since they might not have an internal parallelism in their structure. The MQD encoding is very favorable for parallel compression due to its internal parallelity and simplicity (JGuo et al., 2006). The BCWT algorithm constructs an MQD map in recursive back coding in a pass, which removes all the wavelength trees ahead of the coding (A Said *et al.,* 1996) (DST *et al.*, 2000). See the detailed description of the BCWT in (JGuo*et al.,* 2006).

In this paper, new GPU-based stages have proposed for encoding the wavelet tree-based MQD matrix coefficients with the compact of these steps, parallel multi-pass Qmax search, parallel unit wise encoding, parallel multi-rate Qmin parameter designed for acquiring the progressive results and finally parallel RLE, parallel Huffman entropy coding methods used for building up as parallel backward coding of the wavelet trees (PBCWT), as shown in Figure-3.

PBCWT alters the BCWT by proposing a one-pass encoding system (i.e., all bands have a corresponding MQD matrix) and apply the encoding to enable unit image encoding. The GPGPU model CUDA (Nvidia CUDA) is used in this article. The second part describes the compact of all the steps mentioned above to process the compression method. The third part specifies the comparison-based analysis concerning the compression ratio and the system accelerates between sequential and parallel implementation compression system based on their behavior in the experimental outcomes. Finally, the fourth section directed the future work after the discussion of conclusion.

©2006-2020 Asian Research Publishing Network (ARPN). All rights reserved.

CUDA ARCHITECTURE

GPGPU is the best architecture to speed up parallel algorithms, due to its variation; all devices can easily handle high-resolution images, or perform better performance on video. GPGPU is conventionally designed for graphical tasks, but with the help of different framework platforms, general purpose tasks can be achieved. Parallel Computing Platform Frames (1) NVIDIA Compute Unified Device Architecture (CUDA) and (2) Kharos Group's Open Computing Language (OpenCL) (Khmous et al.). These two general-purpose tasks are supported. Finally, CUDA has chosen CUDA to develop this work because books, online classes, blogs, tutorials, and forums are widely available. The following Figure-1 shows the internal structure of the CUDA with their respective memories, including threads, blocks, and grids.

General Purpose Graphical Processing Unit (GPGPU) does, not only process the graphics-oriented tasks, but also can perform the general-purpose tasks. The GPU is an enhanced computational resource processor. A full-fledged parallel processor has been evolved by adding the fixed and special purpose features to the processor. Initially, we began to perform graphics related applications through the pipeline with a well-suited structure. Eventually, it became a robust architecture, that is, the GPU. Primarily the GPGPU has two programming models: (1) Computation Unified Device Architecture (CUDA) invented by the NVIDIA Corporation and (2) OpenCL developed by the Khronos Group by vendors. Out of two, we have chosen CUDA to implement the parallel stages of this dissertation. The general structure of CUDA has several types memories used to model the architecture as shown in Figure-1.

The different types of memories are (1) global memory, (2) shared memory, (3) constant memory, (4) local memory. Here, the CPU, memory will host the entire task, so can call it as "host memory, " and the GPU memory referred as "device memory," before performing calculations which obtains the required data from the host memory.

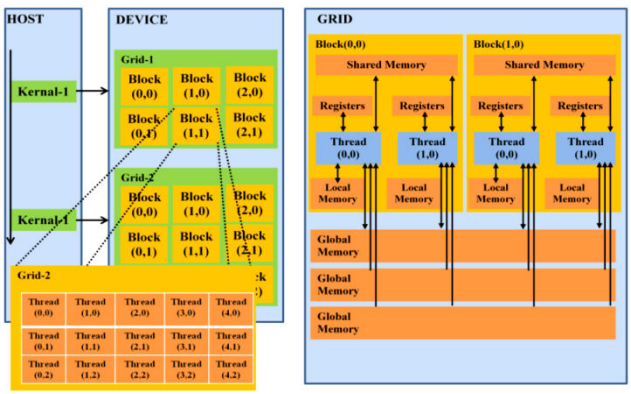

Figure-1. CUDA hierarchical structure.

Usually, among the memories in the architecture, the global memory is huge and as well costly operation too, so as less as possible to access the data from the global memory, which gives more gains, concerning the complexity. If any variables are trying to access the same location or data, which may face the problem called the race condition so the access of global memory should have coalesced.

SEQUENTIAL BACKWARD CODING OF WAVELET TREES

The basic BCWT algorithm proposed by invented by Texas Tech University in 2005 (JGuo et al., 2006), and continually adding the best features into the codec. The best feature QP or Qmin parameter suggested for bit-rate control. Qmin decides the image quality and compression efficiency, whenever Qminis set to zero, the compression system is nearly lossless otherwise lossy.

The BCWT algorithm general block diagram shown in Figure-2. The wavelength tree-based coding algorithms are EZW, SPIHT, and their variants, begin the encoding process from the top level to the bottom level in the wavelet tree. Hence, the wavelength tree should repeatedly be scanned to complete the coding process. Those algorithms consume more resources by examining the tree multiple times, i.e., time, power, memory, computations, etc. The BCWT algorithm will overcome those issues.

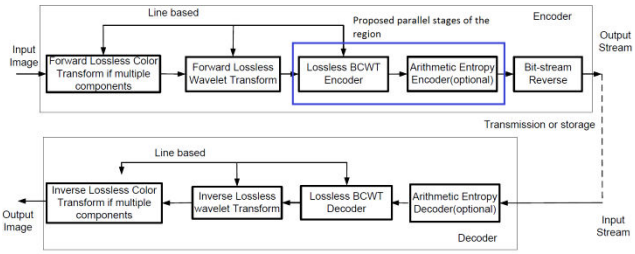

Figure-2. Lossless BCWT block diagram.

The BCWT algorithm is an entirely reversible procedure, which starts the encoding process from the bottom level to the top level in the wavelet tree by scanning single time. The entire encoding procedure in BCWT will be finished in a single backward pass, for this reason, it is incredibly efficient. The BCWT encoding algorithm uses the higher-frequency coefficients after the dyadic partition of the image. Based on these coefficients the MQD and the MQL have built up the BCWT.

The BCWT algorithm 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. $q_{i,j} = \begin{cases} \lfloor \log_2 |c_{i,j}| \rfloor, if |c_{i,j}| \geq 1 \\ -1 \quad , otherwise \end{cases}$: 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).

©2006-2020 Asian Research Publishing Network (ARPN). All rights reserved.

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

q_{min}: The minimum quantization threshold.

$m_{i,j} = \begin{cases} q_{O(i,j)}, if\,(i,j) isin\ the level to sub bands \\ \max\{q_{O(i,j)}, q_{L(i,j)}\},\ otherwise \end{cases}$: 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 $m_{i,j}$- $q_{L(i,j)}$.

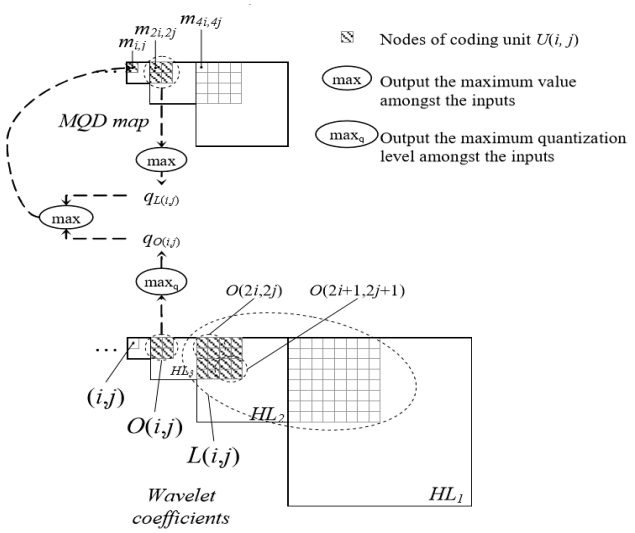

Figure-3. BCWT coding unit procedure.

BCWT coding units have many smaller branches of the wavelet tree of their respective MQD map nodes and have repeatedly been processed. Figure-3 shows the coding unit of U (i, j) in the HL sub-band. It will display intermediate coefficient coordinates. Table-1 list of steps is used to encode the BCWT coding unit U (i, j). Level 3 sub band (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. As a result, the BCWT's MQD map is only 1/15 of size with size, and 1/4 is not usually the size (JGuo*et al.*, 2006).

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

PARALLEL STAGES OF BCWT ALGORITHM

As of now, we have seen the sequential BCWT algorithm implementation. This is the section we are going to be altering the above-discussed stages in parallel. Now all types of stages will design and observed their results individually in the respective papers (Suda*et al.*, 2017, Satya *et al.*, 2017, CSB *et al.,* 2017, Esuda*et al.,* 2017) after discussion of all these stages, we will integrate them and find the final-results. Subsequently, compare these results concerning compression ratio and execution time with the popular algorithms such as JPEG-XR and BCWT. Fig.4 shows significant steps and their data structure will be exploited as the way in the compositions of the compression procedure. Here the GPU called "Device," can be interacted with CPU called "Host," see in Figure-1.

A. Forwarded 2D LDWT

The forward 2D Lifting-scheme DWT (LDWT) produces LDWT coefficients from the input image for further encoding steps. 2D 5/3 LDWT is used to handle lossless compression procedure. In a one-level 2D LDWT forward transformation, the block of horizontal LDWT values is different from the vertical LDWT since this is following the row-major order reading. So, the GPU required access in the way of coalescing.

B. Parallel Multi-Pass Qmax search

The Qmax should get the maximum value of a coefficient with the support of the MQD, which is at the highest level of the MQD matrix and is referred to as “search matrix.” Once found the maximum element, that corresponding four bands at the highest level of the matrix will encode first and can be achieved by performing a reduction operation on the search matrix (Suda*et al.*, 2017). To obtain a Qmax search result requires two steps: (1) Horizontal Reduction and (2) Vertical Reduction.

©2006-2020 Asian Research Publishing Network (ARPN). All rights reserved.

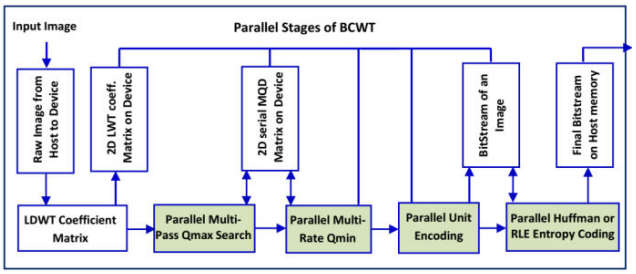

Figure-4. Stages of parallel BCWT.

In vertical reduction, each block received two columns of the matrix from CPU host memory. In each block having two columns of elements, so these columns compared each other, who owns those will be staying in the memory, and other will be removed. Then again, every two blocks will share the column of elements and again, compare and follow the above procedure. It will continue to all blocks, and it has done hierarchically. Finally, only one column will be staying in the memory. This column will pass to the horizontal reduction phase.

In the Horizontal reduction, this column will be assigned to a block. As per the requirement, the GPU threads will be launched in the concerned block to process it. Here, every two threads will compare and stay owned thread to participate in the next level. Finally, this would get a maximum coefficient value (Suda*et al*., 2017).

C. Parallel Multi-Rate Qmin approach

The Qmin parameter determines the total computation and transmission time in BCWT, which effectively chopping the wavelet coefficient bits, thus providing the speed and computational efficiency. In the encoding process, the lossless method usually takes Qmin = 0. Otherwise, the value changes accordingly. Some coefficients with Qmin1 = 0 and another can have Qmin2 = Q0 + 1. This strategy called for a dual quantization level (Suda*et al*., 2017).

$$QI = Q_0 + \frac{C_2}{C}$$

The parameter will change from Qmin1 to Qmin2 in (Suda*et al*., 2017) as the C for all coefficients and C2 as indicated by C. The value of C2 can be estimated when the relationship of QI and BR parameters is known.

The above procedure will partially adapt to achieve the bit-rate control in the following method. Sometimes, the BCWT encode the DWT level, but not by a bitplane, so it's impossible to make progressive quality by incorporating a bit rate in the process. However, it is easy to produce a modifying bit-stream rate without the significant change of the BCWT algorithm. In the BCWT encoding scheme, bits are added temporarily to LIFO (last-in-first-out) buffer because this rate scalability is possible and is called "bit-distributors." Those buffersare adapted to a specific quantized bit plane level. Each bit sent a supplemental size to the corresponding buffer. When the encoding process is completed from the highest, to the highest level, the output will be sentin reverse (Suda*et al.*, 2017).

D. Parallel unit encoding

Parallel encoding is the main component of the encoding phase in which coefficients are encoded in a bitstream (Satya *et al*., 2017). The parallel encoding band is displayed in the unit from the highest to the lowest level and the set of "Unit." Parallel encoding phase "Unit" is the basic unit of encoding and consists of an MQD with four related LDWT coefficients in one band at one level. A thread specifies each unit. Within each group, the encoding order of the units is from the upper-left corner to the lower-right corner done row by row. Because the encoding of those units is enabled simultaneously, their order is not a temporary order, rather than a bitstream produced as a physical location order.

Parallel unit encoding follows:

a) The MQD in each unit is encoded by Qmax first (top level)

b) MQD coefficients are encoded in that subject.

Along with encoded bitstream, the unit is referred to as the second array of "length array." Each unit enters the group-wise encoding.

Group-wise encoding is an excellent way to improve the compression ratio. This process only applies to intragroup elements, not on the inter-group elements. As shown in (Satya *et al.,* 2017), this group has many unused bits so that we can use those bits for another component of the 32-bit string; we can concatenate two-bit strings into one string. This process improves the compression ratio by decreasing bits that are not used in the string and can construct two strings attached as one string. However, this method should be implemented in a series of sequences, depending on each other. This may cause the algorithm to disrupt the acceleration of performance. If multiple threads attempt to access the same element, the situation creates a race condition that may lead to the wrong access.

E. Entropy coding techniques

Parallel RLE encoding

Run-length encoding (RLE) scheme is the most suitable for DICOM images because it has a large redundancy. The RLE handles the RLE, which is the short form of run [0, N-1] run (r << N), where r-run is a subgroup subdivision. The RLE sets (L0, V0), (L1-V1),... (Lr-1; Vr-1), respectively, are the length of the plot of Li and Vi is the symbol of the run that repeats.

RLE is a straightforward compression algorithm. Say we have input data [1,2,2,4,7,7,7,8,9,9]. If we run RLE on this data, we obtain the compressed data as [(1,1), (2,2), (1,4), (3,7), (1,8),(2,9)]. By executing RLE, by executing the RLE, the repetition of the repeated data above the data (x, y), y is repeated by the name of the repetition symbol, and the number x repeats the number. So the pair (3,7) indicates the data [7,7,7].

©2006-2020 Asian Research Publishing Network (ARPN). All rights reserved.

The parallel RLE algorithm 5.1 described as follows (CSB *et al*., 2017). In the implementation, the output pair to be split into two arrays. The first line consists of x: s, and contains the second array y: s. Thus, the input data will run PAR_RLE [1,2,3,6,6,6,5,5] with two output segments [1,2,1,3,1,2] and [1,2,4,7,8,9]. The parallel RLE performed operations on distributed input data in chunks.
Parallel RLE steps:

First Step: Construct a *Mask* array from the input stream.
Second Step: The fastest parallel Blelloch's tree-based algorithm is an inclusive prefix sum scanning approach applied on *Mask* to obtain the *Scanned_Mask* elements.
Third Step: We have to create a Compact_Mask, and it has two special cases
Final Step: The *Scatter_Kernel* procedure had drawn the values of *Symbols_Out*and *Counts_Out* from *Compact_Mask*.

Parallel Huffman entropy coding

Huffman and David proposed a basic method to optimize the redundant data by reconstructing the minimum code word per symbol and which sequentially exploited in $O(n\log n)$ time. Each k^{th} Probability called $P(k)$of N symbols. $\sum_{k=1}^{N} P(k) = 1$, on the length of a message,$L(k)$is the length of the code word of it. Hence, the length of the message on an average is $L_{av} = \sum_{k=1}^{N} P(k)L(k)$. The Huffman entropy coding technique widely used in various applications like image and video compression algorithms and communication protocols where we needed to compress the data.

An implementation of a serial Huffman coding technique is made up of three major steps (Esuda*et al.*, 2017). The first step histogram finds the occurrences of symbol probabilities from the input stream. The second step constructs the Huffman code tree, and the last one generates the binary code word from the generated tree. Each symbol may encode with a variable length bit according to their occurrences in the input stream, whichever comes more frequently that obtains unique shorter codes by assigning for heavily used symbols and larger codes for less-used symbols. Based on the symbol's code word can generate the tree by picking up the minimum pair of input symbols done it's by repetition. Generate code words for each symbol by appending the codes by tree traversals from root to leaf after the assignment of the alternative codes by '0' (for Left Child) and '1' (for Right Child) to the Huffman tree.

These code words were appended to the output stream one after one, other than this scenario not feasible to perform in parallel. After distributing the data as equal sizes to all chunks and they operate the tasks independently because the arrangement of the output stream has done bit by bit as serially.

Parallel stages of algorithm

a) Find gray-level probabilities of symbols of finding the Histogram parallelly on the GPU.
b) Find and combine *two_min* probabilities on the GPU after the construction of a serial Huffman tree on the CPU.
c) Generate code words by tree traversals after the assignment of the alternative codes by '0' (for Left Child) and '1' (for Right Child) on the GPU.
d) Apply a Parallel Prefix Sumto find the offset of a symbol in the generated code word stream on the GPU. The compressed bit stream generated parallelly on the GPU.

So far, we have designed a model that combines all of the stages that we have seen, as shown in Figure-4. This model applies to various types of medical images, and the results are discussed in the next section.

EXPERIMENTAL RESULTS ANALYSIS

The purpose model PBCWT algorithm is to achieve a faster compression when performing parallel ordering algorithms that connects an acceptable compression ratio. The PCWT algorithm is compared to a CPU implementation of the BCWT and JPEG-XR compression algorithm (Srini*et al.*, 2007). The JPEG2000 was not selected to match our algorithm, although it was based on a wavelet transfer due to its high complexity and relatively low compression speed. The JPEG-XR achieved high-speed and competitive lossless compression rate (RVeerla*et al*., 2012).

The experimental PC equips Intel i7 2.93GHz CPU with 8GB memory. The GPU hardware is NVIDIA GTX940 on the same PC. The JPEG-XR compression algorithm is implemented by using the Microsoft .NET 4.0 framework. The PBCWT compression algorithm implemented by using native C++ (host-side) and CUDA C 4.0 (device-side). Program files run on Microsoft Windows 10, 64-bit with the 7th Generation OS. For the experiment, all standard images have been used according to the existed papers usage.

©2006-2020 Asian Research Publishing Network (ARPN). All rights reserved.

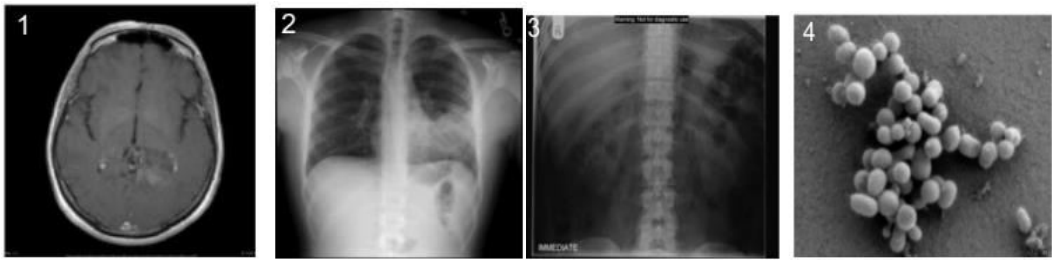

Figure-5. Four sample images.

In the experiment applied four publicly available medical images on three algorithms such as PBCWT, BCWT, and JPEG-XR. In Table-2, listed the results of the test. Four pictures shown in Figure-5.1 to 5.3 are medical images, and Figure-5.4 is an organism image. Each medical image is taken in five different quantitative sizes, with an average of 20 different sizes of four medical images that find the results.

Table-2. Cooperation among PBCWT, BCWT and JPEG-XR.

Medical Images	Memory Range	PBCWT	PBCWT	BCWT	BCWT	JPEG XR	JPEG XR
		CR	RT	CR	RT	CR	RT
Brain	2MB≈5MB	3.588	40.8	3.448	79.8	3.882	105.8
Chest	5MB≈10MB	3.23	53.2	2.714	92.2	3.582	201.6
Spinal	10MB≈15MB	3.144	77	2.802	176	3.398	428.4
Organism	20MB≈40MB	2.418	173	2.092	412	2.656	1117.6

Table-2 shows that PBCWT is better than BCWT and JPEG-XR to process larger images. In the below results the compression ratio will be almost negligible when compared with the JEPXR. The average compression ratio increased from 3% to 4% over BCWT. PBCWT can achieve 4-5 times faster compression speed than the JPEG-XR and PBCWT 2-3 times faster than BCWT. If the image size is more significant than 15MB, the results will improve.

CONCLUSIONS

The PBCWT algorithm works well with remarkable speed and the comparable compression ratio. If we supposed to design all stages of the algorithm in parallel, we could expect the tremendous results on an MQD based coefficient encoding algorithms. As per the results, the PBCWT achieved up 4-5 times faster coding speed reached than a famous JPEG-XR algorithm and with the BCWT algorithm. The scalability of PBCWT does high since most parallel stages not depend on the hardware limits.

The additional amendments to the algorithm made a remarkable performance, including parallel Qmax search stage and another group-wise encoding stage. The PBCWT algorithm is still designed and improved in all kinds of image compression.

REFERENCES

D. A. Clunie. 2000. Lossless compression of grayscale medical images. Proc SPIE.pp. 74-84.

A. Said and W. A Pearlman. 1996. 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): 243-250.

C. Shoba Bindu, Sudarshan. E and Ch. Satyanarayana. A Parallel RLE Entropy Coding Technique for DICOM Images on GPGPU. IEEE Cofer.Proc.2017 (ICCTCEEC 2017), ISBN: 978-1-5386-3242-0, pp. 175-178.

D. A. Clunie. 2000. Lossless compression of grayscale medical images. Proc SPIE.pp. 74-84.

D. F. Schomer, A. A.Elekes, J. D. Haze *et al*. 1998. Introduction to wavelet-based compression of medical images. Radio Graphy. (18): 469-481.

D. Staubman. 2000. High performance, scalable image compression with EBCOT' IEEE Trans. Image Proc.9(7): 1158-1170.

E. Sudarshan, Ch. Satyanarayana and C. Shoba Bindu. 2017. A Compact Parallel Huffman Entropy Coding Technique on GPGPU using CUDA. ARPN Journal of Engineering and Applied Sciences, ISSN:1819-6608.

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

©2006-2020 Asian Research Publishing Network (ARPN). All rights reserved.

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

Khrnous group. The open standard for parallel programming of heterogeneous systems. http://www.khronos.orglopencl/.

M. Che and J. Liang. 2010. GPU Implementation of JPEG XR. Proc. Of SPIE-IS&T Electronic Imaging, SPIE. Vol. 7543,754309.

M. D. Adams and R. Ward. 2001. Wavelet transforms in the JPEG2000 Standard. in the IEEE Pacic Rim Conference on Communications, Computers and Signal Processing.

NVIDIA. CUDA parallel computing platform. http://www.nvidia.com/object/cuda_home_new.html.

NVIDIA. Popular GPU Accelerated Application: http://www.nvidia.com/docs/TO/I23576/nv-aplications-cataloglowres.pdf

Prashanth, H. S., H. L. Shashidhara, and Balasubramanya Murthy KN. 2009. Image scaling comparison using a universal image quality index.In Advances in Computing, Control, & Telecommunication Technologies, 2009. ACT'09. International Conference on, pp. 859-863. IEEE.

R. Le, J.L. Mundy and R. L. Bahar. 2012. High Performance JPEG2K Decoder.in IEEE 23rd International Conference on ApplicationSpecifc Systems, Architectures and Processors.

R. Veerla, Z. Zhang and K. R. Rao. 2012. Advanced image coding and its comparison with various still image codecs. American Journal of Signal Processing 2012. 2(5): 113-121.

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

S. Srinivasan, C. Tu, S. L. Regunathan and G. 1. Sullivan. 2007. HD photo: A new image coding technology for digital photography.in Applications of Digital Image Processing.

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

Sudarshan E., Ch. Satyanarayana and C. Shoba Bindu. 2017. A Secured Model to Reduce the E-Health Record with Lossless BCWT on the Hybrid Cloud. International Journal of Applied Engineering Research, ISSN: 0973-4562, 12(16): 5624-5630.

W. J. Van der Laan, A. C. Jalba and J. B. T. M. Roerdink. 2011. Accelerating Wavelet Lifting on Graphics Hardware Using CUDA. IEEE Transactions on Parallel and Distributed Systems. 22(l): 132-146.