VOL. 12, NO. 24, DECEMBER 2017 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

A COMPACT PARALLEL HUFFMAN ENTROPY CODING TECHNIQUE ON GPGPU USING CUDA

E. Sudarshan[1], Ch. Satyanarayana[2] and C. Shoba Bindu[3]
[1]Jawaharlal Nehru Technological, University, Anantapuram, India
[2]University College of Engineering, Jawaharlal Nehru Technological University, Kakinada, India
[3]Jawaharlal Nehru Technological College of Engineering, Anantapuram, India
E-Mail: medasare@gmail.com

ABSTRACT

Various imaging applications have adaptively used the lossless Huffman entropy coding technique since the redundancy of an image data will be expelled at the precise level. We proposed an accelerated parallel Huffman entropy coding algorithm which implements on the environment of GPGPU using CUDA architecture. This algorithm proceeds with a parallel histogram approach for determining the occurrence of every symbol of an input data, from that we generate a code word for every symbol after the construction of a sequential Huffman tree. Subsequently, the compressed data obtained as in hexadecimal form after applying the adaptive approach where it reads the data parallel as word wise (8, 16, 32, 64 bits) to the code word. The experimental results showed that the GPGPU runs the parallel code with the speed of 46X than the CPU's serial code.

Keywords: parallel huffman, entropy coding technique, **e**lectronic health record, DICOM, CUDA, GPGPU, TRISH, huffman tree.

1. INTRODUCTION

The most significant forthcoming imaging application is an Electronic Health Record (EHR) [1]. EHR is the patient's health care data a framework which gives more clear learning about the patient's illness with the goal that the treatment should be possible in the most suitable both as far as the way and also the economy. The patient's EHR data transferred to the clouds to specialists, academicians, researchers, and others in examining the patient's disease. Every patient's profile (EHR) should refresh after every once in a while for the well-being, for example, a patient's malady, digestion, MRI, CT examine reports, ECG, Thyroid, etc. Those images are called DICOM (Digital Imaging and Communications in Medicine) images [2] [3] [4]. So that, the comparing changes in the treatment can be made precisely according to the patient's physiological changes. It demands to share rapidly among the professionals.

Therefore, the GPGPU accelerates the entropy coding technique with a tremendous speed an account of parallelism. The well known and widely used Huffman entropy coding technique is the most appropriate method for the above said situation by providing the boosting feature under the name of the parallelism where this will be supporting the hardware and software environment.

2. BACKGROUND

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 [5] 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. 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 fewer-used symbols. Based on the symbol's code word can generate the tree by picking up the minimum pair among 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.

Every step in an algorithm converted as a parallel step except the building of the Huffman tree which consumes more time since it is in serial. Many researchers have investigated the Huffman coding algorithm on this topic to convert it as parallel with different aspects.

P.Berman presented an efficient parallel Huffman coding algorithm at a logarithmic time and linear work of maximum code word length H in O(H) with n-processors after sorting the elements. In any circumstances, the height of the Huffman tree satisfies at O(log n) times [6]. Which improves the construction of the Huffman tree in the reduction scheme by a concave least weight subsequence

and it runs at O(√n log n) times, observed by Lawrence and Larmore [25].

Ritesh Patel *et al*. [7], demonstrated a parallel data compression algorithm on GPU through three major parallel approaches (1) Burrows-Wheeler-Transform (BWT), which practices the two level hierarchical sorting schemes. (2) Move-To-Front transform (MTF), which uses a parallel scanning system. (3) Huffman coding, which executes a parallel reduction scheme by achieving the reduction in $\log_2 n - 1$stages, in the bzip2 compression pipeline in the parallel Lossless Data Compression on the GPU environment. Mikael KarlssonRudberg and Lars Wanhammar introduced in [8] a parallel Huffman decoder model, which developed by adopting the pipeline method, where it reduces the symbol decoder requirements, and the proposed architectures run at high speed.

The worthwhile parallelized technology is the GPGPU (General Purpose Graphics Processing Unit) which exploits by the CPU (Central Processing Unit) [**9**]. The GPU is a specialized computing device which rapidly manipulates at the high rate amount of graphical pixels. The GPGPU modeled as different co-processing units by the combination of CPU and GPU, where the application instruction set has the serial portion that usually executes on the CPU, and the rest of the computationally-intensive non-serial portion runs on the GPU.

A new massive parallel architecture CUDA [10] has introduced by NVIDIA where we accelerate computations in parallel on the GPU unit. CUDA is a parallel processing platform and programming model, empowers impressive credits in computing performance by the harness of the GPU [15, 16]. CUDA and GPU create an incredible impression in the world of parallel computations which motivated to pursue this work.

Ana Balevic introduces the Variable-Length Encoding (VLE) is in [11] reduces the input data by replacing the fixed length shorter code words. The GPGPU architecture used to accelerate the process of parallel compression blocks with atomic operations while performing threads writing in the image transformation and motion estimation and achieved speeds up to 35-50X. This algorithm supports the fixed length code word to substitute instead of variable length code words where it restricted to 32bits, so this makes inconvenient to adapt.

In [12], R.L. Cloud, M.L. Curry, *et al.*, presented a Huffman coding technique that achieves the speed up to *3X* by the composition of blocks compression separately in the decompression portion. Rahmani H *et al*. [23] presented a parallel Huffman entropy coder on the NVIDIA CUDA platform, the code word constructed serially for every symbol, later generate an output stream as introducing a byte stream in parallel and achieved 22xspeed than the CPU's serial code. Howard Paul G and Jeffrey Scott Vitter [24], presented an algorithm for high-resolution images using a hierarchical MLP method for Huffman codes or quasi-arithmetic codes and made them as parallel.

In this paper, where we required to perform an entropy coding for an unbounded of image data flawlessly as swift as that we proposed called as a compact parallel Huffman encoding technique. We typically used different lengths of image data to obtain the promising speed over the CPU's speed.

3. PROPOSED METHOD

The Huffman coder has three major stages, initially, generates the histogram bins to find the frequency probability for every symbol in the input stream, the second stage constructs the code word tree. In the third stage, every symbol gets the code word from the tree by applying traversal from root to leaf and finally compress the bit stream as a bitstream form. The third step is the real hurdle due to its variable length code word. During the implementation, every stage can be performed in parallel except the construction of a tree along with the assignment of codes. In the parallel implementation, every symbol obtains a CUDA thread generates the code word by reading the word-wise (8, 16, 32, 64,...) adaptively from the intermediate pixel's code word stream results likewise it performs iteratively until exhausts the input symbols as shown in Fig.5. Every symbol code word position should find appropriately from the resulting code word stream by a thread. These threads have to synchronize properly for the same memory location without fail; this is called a race condition, not only perform concurrent reads, but also carry out the write operation on the property of atomicity to avoid the race condition.

Due to the plethora of internal memory of every thread, the code words can write parallelly onto the global memory heedlessly without effect from synchronization and race condition problems. The memory mentioned problem conquered by the adoption of the parallel prefix sum algorithm, where it finds the byte offset for every symbol to store a code word in the encoded bitstream. In the final step, combining an every consecutive 16bits from the encoded bit stream and is shown as a compressed hexadecimal bit stream and as parallels.

The Huffman entropy coding technique [5] substitutes the symbol by a variable length bit code, where the code word length is the symbol's recurrence of the event.

The proposed parallel computational stages:

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 Sum to find the offset of a symbol in the generated code word stream on the GPU.

e) The compressed bit stream generated parallelly on the GPU.

3.1 The major computational components

a) Find gray-level probabilities of symbols of finding the Histogram parallelly on the GPU.
b) Construction of a Huffman tree with the parallel searching for `two_min` values among the input symbol probabilities.
c) The Prefix Sum procedure

3.1.1 Find the histogram probabilities parallel for every symbol on the GPU

The Histogram technique is the most appropriate for many image processing applications where it requires the compression. This technique determines the frequencies of occurrence of symbols from the given input image and keeps them as in order. The same procedure makes it accelerates by the adoption of parallelism on the suitable platform with an appropriate speed of 50% over the previous GPU methods. Which can achieve by the Threaded Register Interleaved Stride Histogram (TRISH) histogram is a fully parallelized algorithm by avoiding atomic operations and reducing the iterations of execution by improving instruction level parallelism (ILP), thread level parallelism (TLP), and bit-level parallelism (BLP) [13].

The method of Podlozhnyuk's 256-bin histogram [14], the input image pixel's magnitudes fragmented into blocks of equal size. Each fragmented block is assigned to the thread-block to process the sub-histogram separately. The synchronization is not possible between thread-blocks; therefore every thread-block generates the intermediate sub-histogram as a result and have summed up in the next level where it can synchronize (usually in global GPU memory) among the threads to make the final histogram of an image shown in Figure-1.

Every kernel thread-block is consisting an on-chip shared memory to store sub-histogram values. The block of multiple threads may attempt to modify a data concurrently that may cause to arise the race condition problem where the read and write operations are atomic. Hence, the collision-free method like mutual exclusion should adopt to overcome the race condition problem. In the shared memory two kinds of collisions may occur one is inter-warp (between two or more warps) and the second one is intra-warp (within the warp).

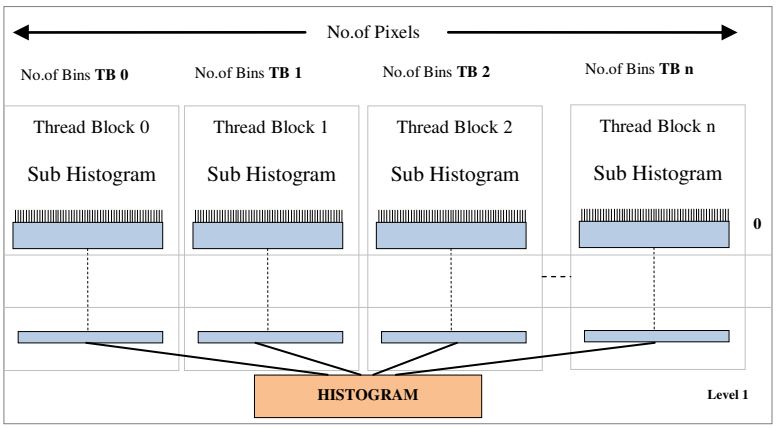

Figure-1. An abstract level of the histogram technique with the children.

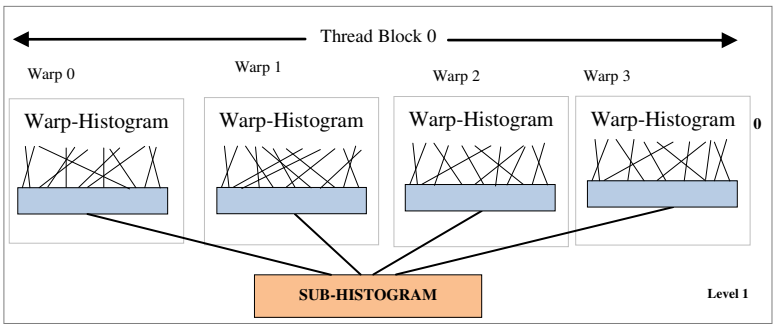

Figure-2. A leaf level scenario of a thread-block.

The Podlozhnyuk's method introduced a partial level of its have histogram called warp-histogram in Figure-2; there will be a synchronization barrier afterward the warp-histograms are summed up as a sub-histogram. There is an alternative solution to overcome the overhead problem where each warp gets one histogram there will be more histograms obtained by dividing the warp magnitudes to threads. Means that every thread will get a

small warp-histogram. TRISH histogram method is faster than the Podlozhnyuk's method by 3.4%. TRISH is the most suitable to our proposed method.

3.1.2 Find and combine two_min probabilities parallelly on the GPU and construct a Huffman coding tree

For the construction of Huffman tree *T* proceed in the bottom-up fashion. If *e1* and *e2* (leaves) are the smallest histogram values in *E*, then a new optimal tree *T'* where *E'=E−{e1, e2} U {e1+e2}* can extend to *E.* The two leaves are emerging into the tree *T* with weights of *e1* and *e2* as a leaf ***x*** of *T'* with a weight of *e1+e2*. The following Huffman algorithm builds with two basic procedures and executes in ***O(n log n)*** time by the adoption of `two_min()` procedure.

1. Algorithm basic_Huffman (E)
Begin
1. **While** |E| > 1 **do**
2. (e1,e2) = two_min(E)
3. e=pair_elements({e1,e2})
4. E=(E-{e1,e2}U{e})
End While
End

The Huffman encoding stage is highly sequential since codes are variable length bits due to its frequency of occurrences. The coding computations are highly dependent on the other codes, which are the issue, not able to partition the computations.

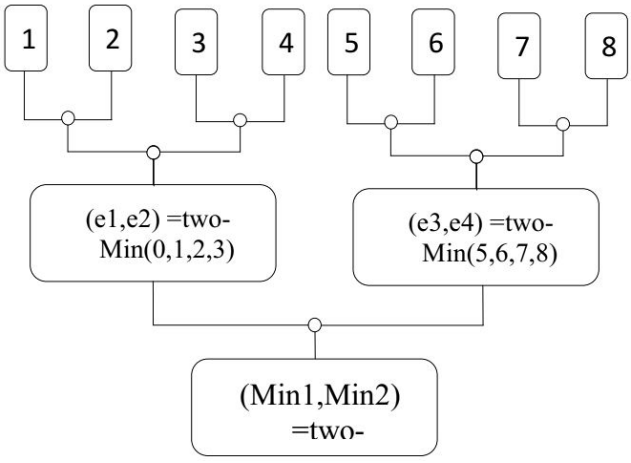

Figure-3. Parallel reductions find the two smallest occurrences.

`two_min(E)` procedure: We perform a parallel search to determine the two_min() elements from the given set *E*. Step2 replaced with a procedure of all_min(E),which returns the min pair *S* from given set *E*. This scenario is called parallel reduction. It obtains the two lowest elements of the given set E, shown in Figure-3. This algorithm executes in *O(log log n)* time per batch with *n* CREW processors [6].

2. Algorithm batch_Huffman (E)
Begin
1.**While** |E| > 1 **do**
2.S= all_min(E)
3. S'=pair_elements(s)
4. E=(E-S U S')
End
End

Huffman contributed an artistic optimal prefix coding algorithm in the year of 1952, which executes at $\boldsymbol{O(nlogn)}$time otherwise, which executes in the linear time whenever the probabilities are in order [5].

3.1.3 The parallel prefix sum procedure on GPU

Every codeword location accurately computed by knowing the length of the codeword to store in the memory, so the codeword bit offset has to calculate by accumulating all codeword lengths which were stored precedently in the bit stream. These codes word locations can be done wisely by using a parallel prefix sum algorithm. The prefix sum algorithm is having a binary associativity operator $\oplus$, where the input array a_0, a_1, ..., a_{n-1} and the generated output array b_0, b_1, ...,b_{n-1} such that b_0=0 and $b_k = a_0 \oplus a_1 \oplus a_2 \oplus ... \oplus a_{k-1}$. Based on the data parallel prefix-sum primitive approach [18, 19], the output bit offset b_k performed by the code word with the lengths a_k, and which allotted to the input symbol.

In Figure-5, the first input symbol 'D' offset is typically is zero; the second symbol 'B' offset is five; the offset of the third symbol 'S' is ten and the rest of the symbols offset's generated in the above manner. The fastest parallel Blelloch's tree-based, inclusive-prefix-sum algorithm is achieved [17, 18]. This parallel prefix sum approach creates up-sweep and down-sweep binary trees to reduce the result explained in the algorithm. 3.3. The Prefix Sum approach uses the binary tree structure with two basic functions up-sweep and down-sweep which perform computations in a manner of parallel. An up-sweep performs the binary tree computations in the way of parallel at each level of the leaf nodes of the root node see in Figure-4. The down-sweep performs the binary tree computations in the way of parallel at each level of the root node to the leaf node see in Figure-4.

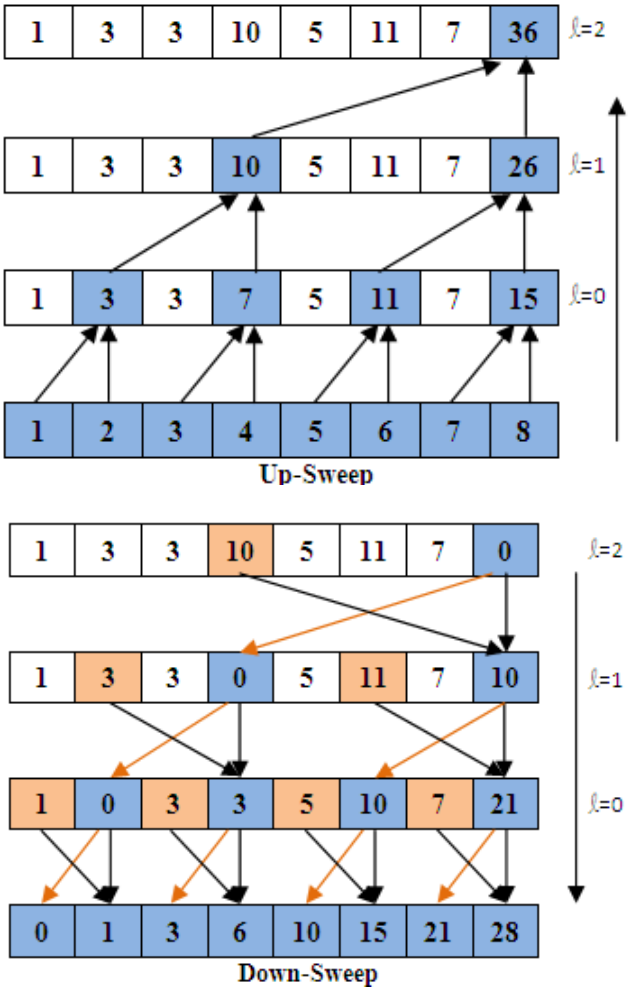

Figure-4. Blelloch parallel prefixsum algorithm.

3. AlgorithmPrefixSumScan(<template>:input[n])
Begin
1. Fori=0 to $lg2^{n-1}$do parallel
2. Forj=0 to n-1 step-up 2^{i+1}do parallel
3. input[j+2^{i+1} -1]=input[j+2^{i-1}]Θinput[j+2^{i+1} -1];
End for
End for
4. input [n-1]=0;
5. Fori=$lg2^{n-1}$to 0 do
6. Forj=0 to n-1 step-up 2^{i+1} do parallel
7. <Template>temp=input [j+2^{i-1}];
8. in [j+2^{i-1}]=input[j+2^{i+1} -1];//left-child
9. input [j+2^{i+1} -1]=temp Θ input[j+ 2^{i+1}-1];//right-child
End for
End for
End

The work complexity of $O(n)$and the depth of a binary tree is $O(logN)$.A further improvement, we can refer a tree-based Blelloch's parallel prefix sum approach is employed in work efficient algorithm which proposed by Harris *et al.* [18] and Sengupta *et al.* [19].

During the generation of the encoding process, each thread is composing a variable length bits the code-word independently for every symbol with an adoption of a depth first search tree traversal approach. All the threads have concurrently written the code-words in the intermediate encoded bit stream without entering into the race condition. The final step of the compressed bit stream, here all threads writing to the global memory independently without overlapping or without a race condition. The parallel tree traversal [21], every thread required to spend $O(n/p + logn)$time using p processors within the EREM model, which delivers with the speed of $p \leq n/logn$.

3.2 Numerical explanation

The CUDA launched a thread to handle a symbol to find the histogram probability in parallel, which used to construct a sequential Huffman tree for creating the code word of every symbol. Finally, the compressed hexadecimal stream has generated after applying the adaptive reading of the code word stream as word-wise (8, 16, 32, 64) as shown in Figure-5. For instance, in *step1* a thread T_0 finds the histogram probability as 0.16 of a color D (Dark), in *step2* this generates a code word `11110` from the Huffman tree. In the *final step*, the thread T_0 reads the 16bit word `1111010100111010` from the code-word stream and this present as a hexadecimal for as `0XF53A` as the way remaining done in parallel.

4. EXPERIMENTAL RESULTS

The parallel Huffman coding algorithm performance evaluation has done by comparing the CPU's sequence coding upon the configuration of NVIDIA GeForce 940MX GPU card and Core 2 Duo processor with the CPU speed of 2.80 GHz is used to run the serial code. We observed the experimental results in the proposed algorithm achieves the acceleration where the data size increases the way speed also increases from 2MB to 32MB as shown in Figure-7. The variable length bits used to encode a symbol code word. The proposed algorithm achieves the speed up to 46% since the GPU pipelines can be worked efficiently for the large volumes of data and this is dominating the entire time of the task.

Therefore, this is the best for the large volumes of data and not appropriate for the smaller ones. If the changes made in the entropy (bits per symbol) the way changes occurred in the speed as linearly varying. In many cases not suggestible to fix the entropy length, especially for DICOM images since it turned to lossy compression. As per Fig.6, if the entropy of the data increased in size along the way the speed will be increased up to 26%, and the speedup will drop up to 21% when the entropy (2, 4, 8, 16) increases.

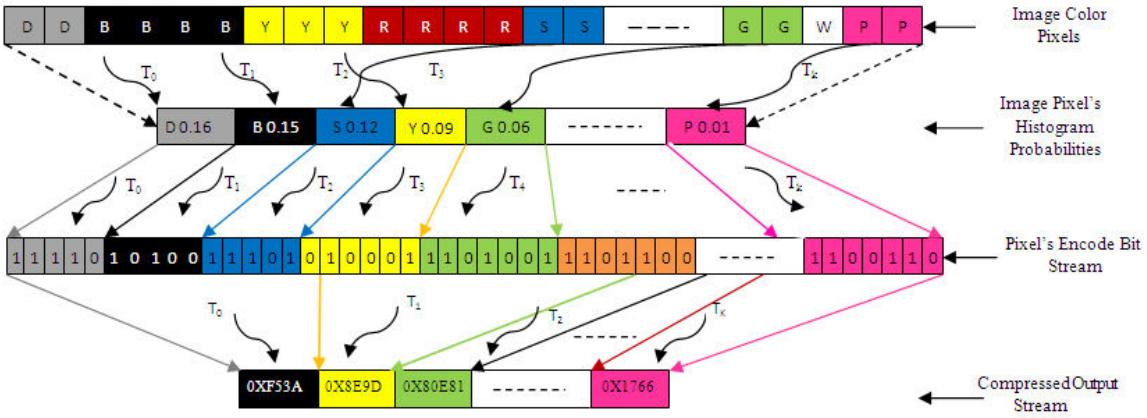

Figure-5. Parallel huffman encoding by word-wise (16 bit) stream.

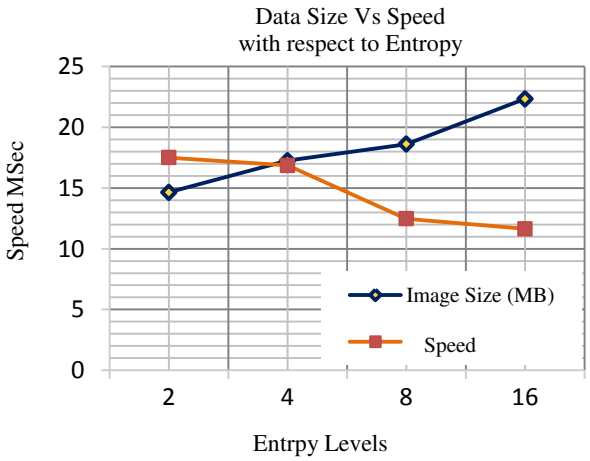

Figure-6. Speedup achieved on the NVIDIA GeForce 940MX compared to with core 2 Duo CPU at a speed of 2.80 GHz as the input data increases.

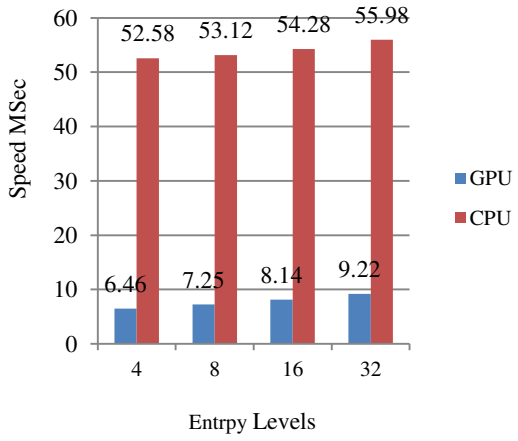

Figure-7. The entropy value increases from 4 to 32 along with the execution time.

Conversely, the DICOM images dealt with large volumes of data typically the entropy bits size also will be getting high. Therefore, the GPU operated smoothly by using synchronization after the distribution of data among the threads.

5. CONCLUSIONS

The parallel implementation of the Huffman code technique on the environment of CUDA and it is experimentally proven that the speed achieved up to 46X over the serial CPU implementation under some considerations. The four procedures computationally major components in the proposed method when it performed parallelly, except the construction of the Huffman tree. The proposed algorithm runs on any type and any size of input data for generating the code word length. The complete parallelized approach performed well with an incredible speed up where it moderates the rush among the levels of threads, registers, and instructions.

Henceforth, this does not only save the internal memory but also compacts the time. The fully parallelized adaptive Huffman coding technique is the most appropriate for the color images where have been creating the skewed binary tree or where the depth of the tree is very high, that we considered as the future work to resolve.

REFERENCES

[1] 2013. History of the Healthcare Information and Management Systems Society (Formerly Hospital Management Systems Society). Healthcare Information and Management Systems Society.

[2] HIMSS, definition EHR, http://www.himss.org/ASP/topics_ehr.asp. DICOM brochure, nema.org.

[3] Kahn CE Jr, Carrino JA, Flynn MJ, Peck DJ, Horii SC. DICOM and radiology: past, present, and future.

Journal of the American College of Radiology 2007; 4:652-657. DOI 10.1016/j.jacr.2007.06.004

[4] Huffman D. 1952. A Method for the Construction of Minimum-Redundancy Codes" (PDF), Proceedings of the IRE. 40 (9): 1098–1101.

[5] Berman P., Karpinski M. and Nekrich Y. 2002. Approximating Huffman codes in parallel. Automata, Languages and Programming. pp. 778-778.

[6] Patel R. A., Zhang Y., Mak J., Davidson A. & Owens J. D. 2012. Parallel lossless data compression on the GPU. pp. 1-9. IEEE.

[7] Rudberg, Mikael Karlsson and Lars Wanhammar. 1997. High speed pipelined parallel Huffman decoding. Circuits and Systems, 1997, ISCAS'97. Proceedings of 1997 IEEE International Symposium on. Vol. 3. IEEE.

[8] Atanasov Dimitar. 2005. General purpose GPU programming. International Conference on Computer Systems and Technologies-CompSysTech.

[9] 2011. NVIDIA Corporation. NVIDIA CUDA computes unified device architecture, programming guide.

[10] Balevic Ana. 2009. Parallel variable-length encoding on GPGPUs. In European Conference on Parallel Processing, pp. 26-35. Springer, Berlin, Heidelberg.

[11] Cloud, Robert Louis, Matthew L. Curry, H. Lee Ward, Anthony Skjellum, and Purushotham Bangalore. 2011. Accelerating lossless data compression with GPUs. ar xiv preprint ar xiv: 1107.1525.

[12] Brown Shawn and Jack Snoeyink. 2012. Modestly faster histogram computations on GPUs. Innovative Parallel Computing (InPar).

[13] Podlozhnyuk V. 2007. Histogram calculation in CUDA,Technical Report, Nvidia, URL:

[14] http://developer.download.nvidia.com/compute/cuda!II/Website/projects/histogram256/doc/histogram.pdf

[15] NVIDIA Corporation Technical Staff. NVIDIA CUDA programmingguide 5.0. Technical report, http://docs.nvidia.com/cuda/cuda-cprogramming-guide.

[16] E. Lindholm, J. Nickolls, S. Oberman, J. Montrym. 2008. NVIDIA Tesla: Aunified graphics and computing architecture," IEEE Micro, vol. 28, no.2, pp. 39-55, 2008.

[17] Guy E. Blelloch. 1990. Prefix Sums and Their Applications. In John H. Reif (Ed.), Synthesis of Parallel Algorithms, Morgan Kaufmann.

[18] Mark Harris, Shubhabrata Sengupta and John D. Owens. 2007. Parallel prefix sum (scan) with Cuda. In Hubert Nguyen, editor, GPU Gems 3. Addison Wesley.

[19] Harris, Mark, Shubhabrata Sengupta and John D. Owens. 2007. Parallel prefix sum (scan) with CUDA. GPU Gems 3.39: 851-876.https://classroom.udacity.com/courses/cs344.

[20] Shubhabrata Sengupta, Mark Harris, Yao Zhang and John D. Owens. Scan primitives for GPU computing. In Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware, GH '07 pages 97_106.

[21] Chen Calvin C-Y., Sajal K. Das and Selim G. Akl. 1991. A unified approach to parallel depth-first traversals of general trees. Information Processing Letters. 38.1: 49-55.

[22] Atallah, Mikhail J., et al. 1989. Constructing trees in parallel. Proceedings of the first annual ACM symposium on Parallel algorithms and architectures. ACM.

[23] Rahmani H, Topal C, Akinlar C. 2014. A parallel Huffman coder on the CUDA architecture. InVisual Communications and Image Processing Conference, 2014 IEEE Dec 7 (pp. 311-314). IEEE.

[24] Howard Paul G. and Jeffrey Scott Vitter. 1992. Parallel lossless image compression using Huffman and arithmetic coding. In Data Compression Conference, 1992. DCC'92. pp. 299-308. IEEE.

[25] Larmore Lawrence L. and Teresa M. Przytycka. 1995. Constructing Huffman trees in parallel. SIAM Journal on Computing. 24(6): 1163-1169.