International Journal of Engineering Research and Development
e-ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 11, Issue 02 (February 2015), PP.01-07

A Review on Image Compression in Parallel using CUDA

Bhavesh Vekariya[1], Chhaya Patel[2]
[1] *PG Student, Department of Computer Engineering, RK University, Rajkot.*
[2] *Assistant Professor, Department of Computer Engineering, RK University, Rajkot.*

Abstract:- Now a days images are prodigiously and sizably voluminous in size. So, this size is not facilely fits in applications. For that image compression is require. Image Compression algorithms are more resource conserving. It takes more time to consummate the task of compression. Utilizing Parallel implementation of the compression algorithm this quandary can be overcome. CUDA (Compute Unified Device Architecture) Provides parallel execution for algorithm utilizing the multi-threading. CUDA is NVIDIA`s parallel computing platform. CUDA uses GPU (Graphical Processing Unit) for the parallel execution. GPU have the number of the cores for parallel execution support. Image compression can additionally implemented in parallel utilizing CUDA. There are number of algorithms for image compression. Among them DWT (Discrete Wavelet Transform) is best suited for parallel implementation due to its more mathematical calculation and good compression result compare to other methods. In this paper included different parallel techniques for image compression. With the actualizing this image compression algorithm over the GPU utilizing CUDA it will perform the operations in parallel. In this way, vast diminish in processing time is conceivable. Furthermore it is conceivable to enhance the execution of image compression algorithms.

Keywords:- GPU, CUDA, Multicore, Shared Memory, Multithreading, DCT, DWT, BWT, EBCOT, MTF, DAST, JPEG2000.

I. INTRODUCTION

Image compression process is resource conservative and time consuming process because it requires large number of floating point computation. To reduce the performance time of image compression algorithm parallel execution is important. There are the many techniques for compression of image. Every uses different type of image encoding and different image compression methods. Compression ratio, Execution time and image quality is varying by method to method. Important parameter like time required to image compression can be reduce by implementing compression technique in parallel. Using CUDA it is possible to implement and execute image compression in parallel. CUDA will accelerate the execution at the maximum wrap on the GPU. Speed up can be possible by making the different part of the algorithm in parallel. Image compression is of two type lossy and lossless. Lossy compression is not giving back the precise unique picture from the compressed image, while loss less compression has the capacity to recover the original image without the any loss of data.

II. LITERATURE REVIEW

Different methods for image compression in parallel utilizing GPU with lossy and lossless mode are enhanced as further ahead in paper.

A. DCT (Discrete Cosine Transform) using Cordic based Loeffler [1]

This DCT is four stage process. Data dependency is involved during every process so every stage need to execute in serial. Still calculation part can be parallelized. Algorithm can be split in two parts: one part is for even coefficient and second one is for the odd coefficients. Even part is 4-points DCT. This part is further separated in next step. For the different density of picture it gives different result of performance parameters.

B. DCT in parallel [8]

The DCT is a separable transform so it is first applied to the column of the 8x8 matrix, and then DCT is applied to the rows of the results. To avoid calculating cosine values during execution, the cosines of different phases are pre-calculated and stores into an 8x8 constant float matrix C. To perform DCT on the column of the macro block M, 8x8 constant matrix is cross multiply with the 8x8 macro block as shown in equation

$$C \times M = DCT\_R$$

Then the second pass of DCT is performed to the row of column DCT result DCT_R as shown in equations:

$$\text{DCT\_R x } C^T = DCT\_2D$$

There are two type of DCT implementation approach. In first one 64 threads for each thread block. From the raster each thread will load one pixel. Based on thread X and Y index read is assigned to each thread. This data is stored in shared memory. Every thread will calculate one element of the output matrix. Then the result is written back its result on the global memory. Second approach utilizes 512 threads for each block. Here each thread block is assigned to process 8 macro blocks.

Quantization is simple calculation set. The image is divided into strides which equals to the nearest 16th divisor. Then the Quantization matrix is multiplied to the image width by width. Quantization matrix below is used as a multiplier/divisor.

There are three implementation are there. In first thread are multiplied and divided to quantize the image but the image is still a part of global memory. In the second image is divided into 8x8 chunks and this matrix grid was quantized using the quantized matrix then written back to global memory. In third implementation used shared memory was loaded with the division of the global image and the quantized values and then the shared copy was used to multiply back the quantized matrix

For all three results, the runtime does not increase linearly with image size. The overhead in large image could be caused by contention to the shared floating point unit, as the DCT computation is floating point intensive. It is unlikely that non-linearity is resulted from limitation in global memory bandwidth as the image size is relatively small compares available global memory bandwidth.

C. JPEG2000 with DWT and EBCOT Tier-1 in parallel [2]

DWT is a sub-band transform which transforms images from the spatial domain to frequency domain. DWT can proficiently abuse the spatial relationship between pixels in a picture.

JPEG2000 processes color components independently, a multi-color transformation (MCT) step is required for remove dependency between the components before performing the DWT so, it implementation on parallel become easy. After this DWT coefficients are partitioned into individual code blocks and coded by the EBCOT independently.

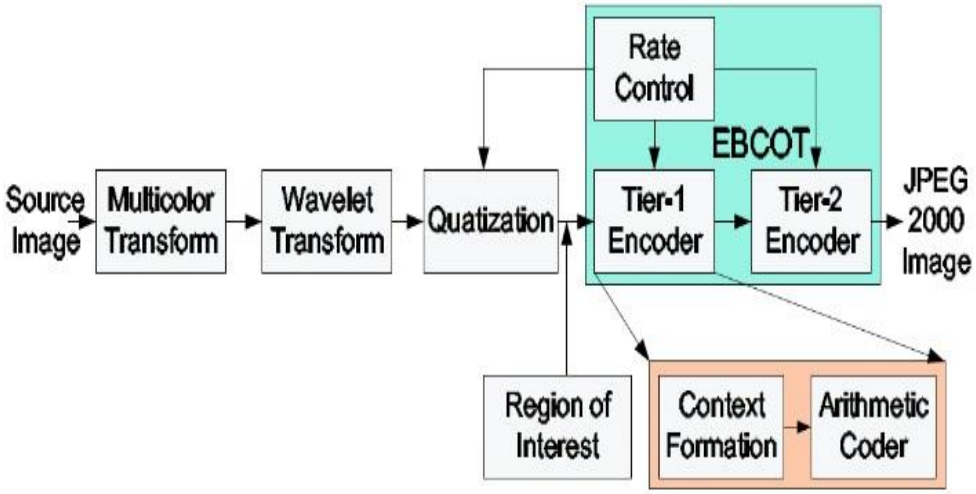

Fig.1: Jpeg2000 Encoding Flow

There are the two-tiers in EBCOT: Tier-1 is for the BPC (Bit Plane Coding) and context adaptive AE (Arithmetic Encoding), Tier-2 is for rate-distortion optimization and bit stream layer formation. Parallel BPC (Bit Plane Coder) is used for remove the inter-sample dependency. The bit plane coder processes code blocks bit plane by bit plane, from most significant bit (MSB) to least significant bit (LSB). In every bit plane, the bit lines are further grouped into stripes of four lines and BPC scans stripes in columns, from left to right using three non-overlap coding passes.

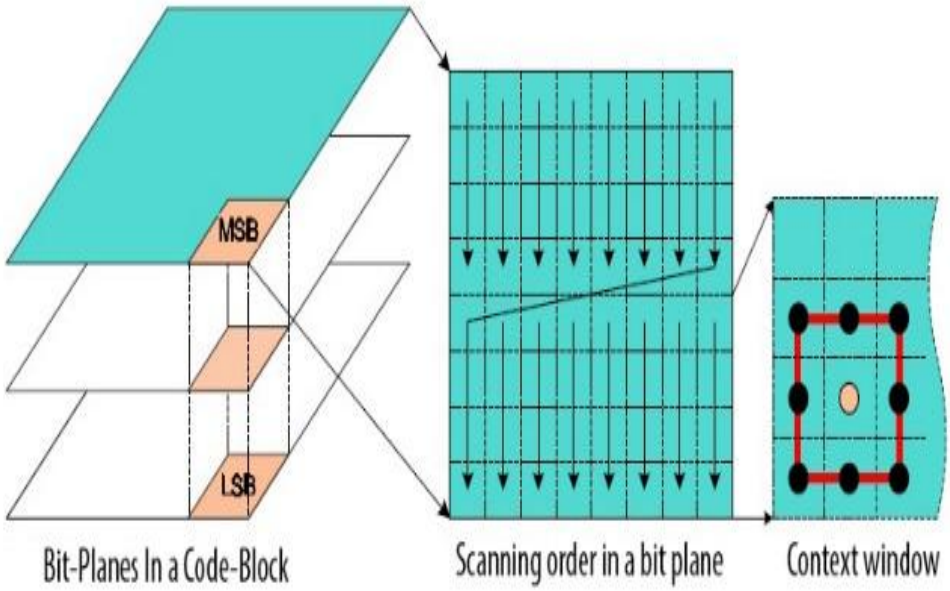

Fig.2: Context formation process of BPC in JPEG2000

Samples are divided in to sub-arrays, the visited sample which are already scanned by BPC also known as pre arrays. And state arrays after BPC finishes coding is called post arrays. This enable the removal of the inter-sample dependency by combining the pre and post state arrays.

In BPC needs state information for 3 x 3 sample but the state variables of each sample are not predetermined since they are updated on the fly. This situation can be considered as a read before update race condition. Therefore, a prediction method is needed to pre-determine the state information before the BPC starts coding so that it can process all the samples concurrently. Parallel reduction steps are performed for prediction. A number of code blocks are processed independently and each code block samples should be processed in parallel using the state prediction method.

In GPGPUs any flow control instruction (i.e. if, switch...) can significantly affect the instruction throughput by causing threads of the same parallel thread block to diverge. The context decision rules are predefined so look-up-tables (LUT) for context formation can be constructed to avoid the branching control flow. Critical to optimize memory usage and memory conflicts are occurs in BPC. Both the memory conflict rate and memory latency can be dramatically reduced with very fast, multi-way shared memory that resides locally on chip.

After the data sets are efficiently allocated into selected memory regions, the utilization of memory, especially shared memory, should be minimized to increase the multiprocessor occupancy of the GPGPUs. Large code block significantly decreases the multiprocessor occupancy which results in significant speedup drop and small size is an impractical for compression efficiency. By using external operation modes parallel Tier-1 coder can handle 64 x 64 code block using small shared memory buffer.

D. bzip2 [3]

Bzip2 is the combination of the three different methods Burrows-Wheeler transform (BWT), the move-to-front transform (MTF), and Huffman coding. BWT compress the input sting and gives equal size transformed string; MTF gives array of indices; Huffman stage gives Huffman tree and Huffman encoding of the list. Actual Compression is done by the Huffman Coding.

1) BWT (Burrows Wheeler Transform): BWT is not itself not performing itself compression. Instead it make easier compression by reversible reordering of string. The new string produced by BWT tends to have many runs of repeated characters, which bodes well for compression.

2) BWT with Merge Sort: In GPU implementation of BWT, Leverage a string sorting algorithm based on merge sort. The algorithm uses a hierarchical approach to maximize parallelism during each stage. Starting at the most fine-grained level, each thread sorts 8 elements with bitonic sort.

Next, the algorithm merges the sorted sequences within each block. In the last stage, a global inter-block merge is performed.

3) MTF (Move to Front Transform): For parallel execution of MTF it is divided in two parts:

- Substring of Main string is used and partial MTF list is generated, that will computes the partial MTF for substring that only contains the characters that appear in substring.
- Two partial MTF lists for two adjacent substring is combined and generate partial MTF list the represents the concatenations of the two substring.

Combine these two methods in parallel divide and conquer implementation. Divide the input string into small 64-character substring and assign each substring to a thread. Then compute a partial. MTF list for each substring per thread, recursively merge those partial MTF lists together to form the final MTF list.

4) Huffman coding: Huffman algorithm replaces each input symbol with a variable-length bit code. Length of code is calculated by symbol`s occurrence frequency.

There are three main computational stages in Huffman codding:

- Generating the character histogram
- Building the Huffman tree
- Encoding data

In Huffman nodes are added and joined, the histogram changes due to that cannot create node in parallel. Instead perform search for the two least-frequent histogram entries in parallel using reduction scheme. bzip2 has an average compress rate of 7:72MB/s over all ,while GPU implementation is 2.78slower with a compress rate of 2:77MB/s.

Lossless data compression focusing on large scale redundancies as in the case of bzip2. The compression methods is quite simple, mostly focusing on only storing deltas between floating point values and removing un-necessary zeroes form the result [4].

E. DAST (Discrete Anamorphic Stretch Transform) [5]

In DAST image is represented by two dimensionally discrete spatial variables n and m by B[n,m]. It is first passed through the DAST and then is uniformly re-sampled (down-sampled) at a rate below the Nyquist rate of original image. To recover the original image form the compressed one, the compressed image is first up-sampled and then inverse DAST is applied to recover the original image.

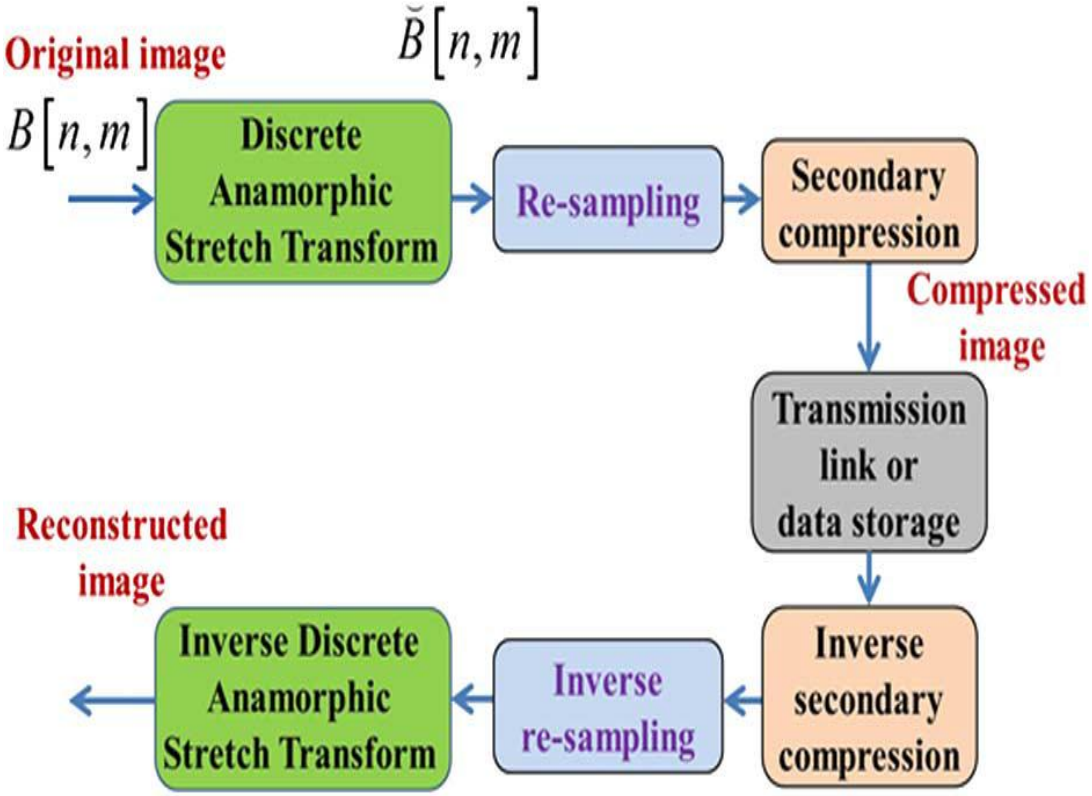

Fig. 3: Discrete Anamorphic Stretch Transform

This method warps the image such that as the increases the image spatial size proportional bandwidth is reduced. This increases the spatial coherence and reduces the amount of data needed to represent the image. Mathematically, DAST is defined as follows:

$$\check{B}[n,m] = \left| \sum_{k1,k2=-\infty}^{\infty} K[n-k1, m-k2] \cdot B[k1,k2] \right|^{N}$$

Where || is the absolute operator. For DAST operation, the original image is convolved with DAST Kernel K[n,m], and then the N-th power magnitude of the result is computed. Here case of N=1 The Kernel K[n,m] is described by a nonlinear phase operation,

$$K[n,m] = e^{j \cdot \Phi[n,m]}$$

F. JPEG2000 with 2D Wavelet Transform [6]

The basic idea of the wavelet transform is to represent any arbitrary function f as a weighted sum of functions, referred to as wavelets. Each wavelet is obtained from a mother wavelet function by conveniently scaling and translating it. The result is equivalent to decomposing f into different scale levels (or layers), where each level is then further decomposed with a resolution adapted to that level.

In this method first image is read in page-locked memory buffer, this main memory buffer is copied to a global memory buffer, 1D-FWT is applied to each row of the image, now for applying again 1D-FWT on each column of the matrix transformation is applied, further transformation is applied to recover the initial data layout. Finally, the resulting image is copied back from the global memory buffer to the page-locked main memory buffer.

G. JPEG2000 with DWT and EBCOT (Embedded Block Coding with Optimal Truncation) [7]

In DWT with EBCOT during encoding an image data is prepared for compression in pre-processing phase first, then entropy-encoded using embedded Block Coding with optimal Truncation (EBCOT) algorithm. Decoding is the reverse process of the encoding. One optional process known as color transform is applied on data before DWT applied, then Discrete Wavelet Transform (DWT) and quantization is applied.

JPEG2000`s main part is DWT and it requires large computations. So, it requires the more number of resources to perform DWT. Lifting scheme is helpful to reduce memory and computational complexity. Lifting plan considers set up information control also diminishes memory conditions.

Lifting scheme analysis proceeds as follows. An input signal is split into even and odd sub sequences denoted as $\{S_i^0\}$ and $\{d_i^0\}$ respectively. These values are further modified using alternating prediction (denoted as p) and update (denoted as u) steps. In the prediction step, the algorithm takes an odd sample in a turn and subtracts a linear combination of its (even) neighbour from it; a prediction error $\{S_i^1\}$ is formed:

$$d_i^1 = d_i^0(s_i^0 + s_{i+1}^0)$$

In the update step, a linear combination of already modified adjacent odd samples is added to each even sample and updated even sequence $\{S_i^1\}$ is formed:

$$s_i^1 = s_i^0 + u(d_{i-1}^1 + d_i^1)$$

2D signals are usually transformed in both dimensions. 1D DWT transform is 1st applied to all rows then to all columns resulting in four sub bands LL, HL, LH, and HH. The LL sub band is an approximation of the original signal and can be further transformed recursively.

The key piece of this DWT is the configuration how to divide the work between string blocks keeping in mind the end goal to give greatest utilization of the GPU. Also size and shape of thread blocks needs to be determined. Because the lifting scheme algorithm alternately works with even and odd samples, an efficient approach is to have one even and one odd data sample per each thread in a block.

H. Lossless Image compression using LS (least squares) adapted prediction [9]

In this method there are five blocks:

- Edge detector
- LS-based adaptive predictor
- Error compensation mechanism
- Entropy coder
- The block of GPUs for the construction of normal equations

Only the normal equation is calculated on the GPU, and the detection of an edge and finding the solution of the normal equation are performed on the CPU.

GPUs is used for the construction of normal equations in an LS-adapted predictive coding system. So that the runtime performance can be accelerated. With the highly increased capability on parallel processing,

matrices multiplication can be carried out very efficiently on GPUs. The main storage, on the GPUs has larger memory access latency. In the implementation an optimized memory access sequence so that the memory pointer does not have to move around the whole memory area between each memory access. Improvement on runtime performance (up to 5 times faster) than that of on the CPU, can be achieved using this implementation.

Table I: Comparison of image compression algorithms

Parallel Method Name	Performance
DCT	20x Speed up
DWT	10x to 20x Speed up
DWT with TIER-1	16x speed up
Bzip2	2.78x time Slower
LS	21.7x (CUDA) and 20.8x (Open MP) speed up

III. CONCLUSIONS

In the compression process of an image required resources and performance time is higher but implementing it on parallel platform of CUDA, it is possible to reduce the compression time. CUDA runs the parallel code over the GPU which has a hundred number of the cores. Using this cores highly parallel execution can be achieved. Up to now many of the methods of image compression are implemented in parallel, there is also scope to optimize further this methods over CUDA to get better result in time and quality of the image.

In DCT data reliance is high because of that parallel usage of this strategy will make the waiting condition. Which lead calculation to hold up until before obliged undertaking ought to be finished. At last this will make delay in execution time [1]. By separating the transform it is possible to achieve desired target [8]. DWT requires color transformation for remove the dependencies which make parallel execution more affordable. After performing the DWT Quantization and EBCOT are performed. EBCOT is processes the data in two Tiers. Parallel implementation is uses the shared memory for data processing. Access of shared memory will increase the performance efficiency of the algorithm. DWT with Tier-1 parallel execution on GPU gives the 16x performance speedup compare to jasper software implementation which runs on the CPU [2]. Fast 2D fast wavelet transform in parallel using CUDA-enabled devices gives up to 21.7 time speed up for same sequential implementation, And using OpenMP similar implementation gives 20.8 times speed up[6]. Parallel DWT with lifting scheme using CUDA is two time faster than same code of jasper in single thread system execution over the CPU [7]. bzip2 is with the run length encoding and multiple table Huffman coding so it will give better compression ratio. Algorithm execution is becomes slower due to this reason. Overall performance is 2.78x slower, but this will enables the GPU to become a compression coprocessor [3]. DAST pre-compression has more than twice compression factor as compare to JPEG2000 implementation. In this method pre-compression as well as PSNR (Peak Signal to Noise Ratio) both are improved [5]. In LS (Least Squares) use GPUs for the construction of normal equations so that the runtime performance can be accelerated. Due to high capability of parallel processing, matrices multiplication can be carried out very efficiently on GPUs. The main storage, i.e., Global memory, on the GPUs has larger memory access latency so optimized memory sequence is used for pointer have not to around the whole memory area between each memory access. This implementation gives up to 5 times faster than same of CPU implementation [9].

REFERENCES

[1]. Kgotlaetsile Mathews Modieginyane, Zenzo Polite Ncube, and Naison Gasela, "CUDA based performance evaluation of the computational efficiency of the DCT image compression technique on both the CPU and GPU", Advanced Computing: An International Journal (ACIJ), Vol.4, No.3, May 2013.

[2]. Roto Le, Iris R. Bahar, Joseph L. Mundy, "A Novel Parallel Tier-1 Coder for JPEG2000 using GPUs", IEEE, 2011.

[3]. Ritesh A. Patel, Yao Zhang, Jason Mak, Andrew Davidson, John D. Owens, "Parallel Lossless Data Compression on the GPU", IEEE, 2012.

[4]. Axel Eirola, "Lossless data compression on GPGPU architectures", 2011.

[5]. Mohammad H. Asghari, "Discrete Anamorphic Transform for Image Compression", IEEE Signal Processing Letters, VOL. 21, NO. 7, JULY 2014.

[6]. Joaquín Franco, Gregorio Bernabé, Juan Fernández and Manuel E. Acacio, "A Parallel Implementation of the 2D Wavelet Transform Using CUDA", 1066-6192/09 IEEE 2009.

[7]. Jiri Matela, "GPU-Based DWT Acceleration for JPEG2000", Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, 2009.
[8]. R Pranit Patel, Jeff Wong, Manisha Tatikonda, and Jarek Marczewski, "JPEG Compression Algorithm Using CUDA".
[9]. Lih-Jen Kau, Chih-Shen Chen, "Speeding up the Runtime Performance for Lossless Image Coding on GPUs with CUDA", IEEE 2013.