Performance Evaluation of Discrete Wavelet Transform Based on Image Compression Technique on Both CPU and GPU

Hussein M. Magboub, and Mahmoud A. Osman

***Abstract*---**This paper investigates the use of the Compute Unified Device Architecture (CUDA) programming model to implement Discrete Wavelet Transform (DWT) based algorithm for efficient image compression. The PSNR (Peak Signal to Noise Ratio) is used to evaluate image reconstruction quality in this paper. The results are presented and discussed.

Key words: DWT, GPU, CUDA, CPU

I. Introduction

IMAGE compression is minimizing the size in bytes of a graphics file without degrading the quality of the image to an unacceptable level. The reduction in file size allows more images to be stored in a given amount of disk or memory space. It also reduces the time required for images to be sent over the Internet or downloaded from Web pages.

A text file or program can be compressed without the introduction of errors, but only up to a certain extent. This is called lossless compression. Beyond this point, errors are introduced. In text and program files, it is crucial that compression be lossless because a single error can seriously damage the meaning of a text file, or cause a program not to run. In image compression, a small loss in quality is usually not noticeable. There is no "critical point" up to which compression works perfectly, but beyond which it becomes impossible. When there is some tolerance for loss, the compression factor can be greater than it can when there is no loss tolerance. For this reason, graphic images can be compressed more than text files or programs.[1]

II. Wavelet Image Compression/Decompression Algorithm

Image compression algorithm follows the following algorithmic steps:

Hussein M. Magboub is with the Computer Engineering Department, University of Tripoli, Tripoli, LIBYA.

Mahmoud A. Osman is with the Computer Engineering Department, University of Tripoli, Tripoli, LIBYA.

1. Read an RGB image matrix (to be compressed).
2. Convert RGB image to YC_bC_r format image.
3. Evaluate the average of each of Y, C_b , and C_r matrices, and subtract each average value from its corresponding matrix. This step uniformly distributes positive and negative values around zero.
4. Zero pad resultant matrices of step 3 to satisfy powers of two dimensions.
5. Down sample both C_b & C_r matrices by a factor of 2 in both column and row directions.
6. Apply DWT to each of the sub sampled matrices individually, wavelet decomposition stops when the length of data becomes less than or equal to the length of the wavelet filter. The used wavelet is determined by the user and the transformation result is stored in the same matrix. A number that identifies the used wavelet is saved along with the compressed file of the last stage [2],[3].
7. Quantize resultant coefficients and apply SPIHT encoding to each of the matrices individually to produce the bit stream that represents it [7].
8. The compressed image final form is a file that contains a header part containing: the dimension of the original image, image resolution, the number of levels used in DWT, a flag referring to the wavelet used for DWT, the average values of Y, C_b, C_r matrices, and the value of the first and last threshold values (n and ln) used in SPIHT encoding for each of the Y Cb Cr matrices followed by the bit stream representing Y,C_b, and C_r matrices .

Image decompression algorithm involves applying IDWT to the resultant bit stream of the compressed file by implementing the inverse of the above stages in a reverse order and making use of the information contained into its header:

1. Read in the compressed image file and apply SPIHT decoding to the bit stream of the compressed file to generate the Y, C_b, C_r matrix using n and ln values calculated by the encoder and stored in the compressed file header. Each of the Y C_b C_r matrices has different n and ln
2. Apply 2D IDWT (inverse DWT) using the same parameters used by DWT. Column and row filtering are

the same. Interpolation is up sampling followed by filtering, it has two parameters, the first is the input data and the other is the filter. Hc and Gc are the synthesis low pass and high pass filters respectively associated with the used wavelet.

3. Up sample both C_b & C_r matrices by a factor of 2.
4. Resize the matrices by removing the padded pixels introduced by the compression algorithm.
5. Add back the average values evaluated in step 3 of the compression algorithm, to the corresponding matrices.
6. Convert YC_bC_r image to RGB image which is used to generate a BMP file representing the decompressed image final form.

II. Results

The 2D DWT based compression algorithm is tested using Baboon 512_512 gray level BMP type images. The DWT is implemented using the CDF 9/7, Coflet 1, Daubechies 2, Spline 3.3, and Symlet 21 Wavelets.

Performance results based on wavelet used is shown in Figures 1, which plots the PSNR versus the bit rate (bpp) using the four wavelets mentioned above.

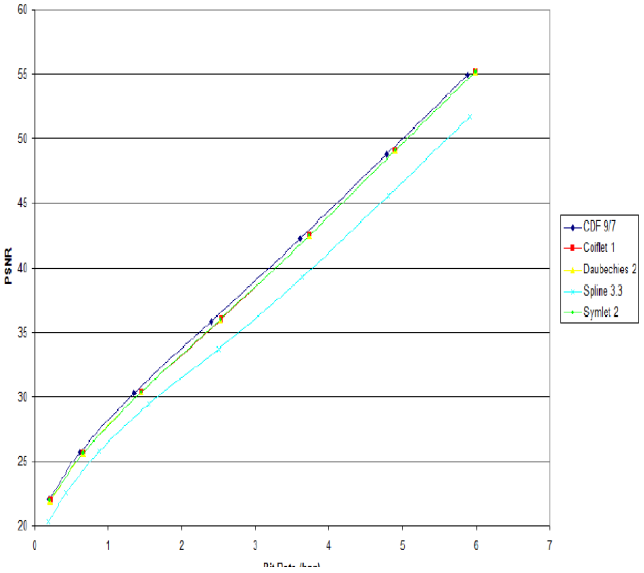

Fig. 1 Comparison Between Different Wavelets for Baboon Image

From Fig. 1, it appears that using CDF 9/7 wavelet results in the highest PSNR out of the wavelets used. However, the Spline 3.3 wavelet results in the lowest PSNR while the other three wavelets provided very close performance.

III. Assisting Cpu For 2d Dwt Using Gpu

Compression of an image (especially very large images) can be tedious on a CPU which normally might have other processes to run along with the compression process. Recently the Graphical Processing Unit (GPU), which was initially used for graphical rendering, has seen a big evolution in its architecture; it has also seen wider usage in general purpose processing [3], [4]. A GPU is a set of multi- processors which give the power to process data in parallel. Since many image processing techniques have sections which consist of a common computation over many pixels, this fact makes image processing in general a prime topic for acceleration on the GPU. The evolution of the new graphic cards led to an increase in the capacities of per-pixel and texturing operations greatly, the reason behind this increase is that the GPU is described as a Single Instruction Multiple Data (SIMD) parallel processor or a streaming processor [5]. The Graphics processors are designed to perform a block of operations on groups of vertexes or pixels, and they do this very efficiently. Programming the GPU version of required algorithms is a straight forward process using computational unified device architecture. For many years ago the wavelet transform process for real-time graphics synthesis is done more slowly on a CPU. However, the current generation of graphics cards has the power, programmability, and floating point precision required to perform this process efficiently, which means the large amount of inherent parallelism and memory bandwidth are exploited to perform the fast transform.

To compare the execution time enhancement of the GPU over CPU, the CDF 9/7 wavelet is used for 2D discrete wavelet transform applied for rectangular matrix using three levels. The computational efficiency is analyzed and evaluated under both the GPU NVIDIA GeForce 8800 GTS using CUDA [8], and a CPU Intel(R) Core2Duo 2.13 GHz with 2 MB L2 cache and 2GB DDR2 installed. TABLE I lists the specifications of the CPU and the GPU used. Both forward (FWT) and inverse (IWT) wavelet transforms are programmed using these processors. The code program compiled using visual studio 2005, and the NVIDIA toolkit 1.1 driver installed (this toolkit driver must be installed for G80 series products). The execution time is tabulated in TABLE II and TABLE III for different image sizes using both FWT and IWT [8], [9], [10].

TABLE I
SPECIFICATIONS OF THE USED GPU AND CPU

Property	GeForce 8800 GTS	Intel Core2Duo 2.13 Ghz
Memory (MB)	640 MB	2 GB
Core clock	500 MHz	2.13 GHz
Processors	96 Stream Processors	Dual Core(2 CPU's)
Memory Bandwidth (GB/s)	64 GB/s	8.5 GB/s

TABLE II
FORWARD WAVELET TRANSFORM EXECUTION TIME

size	CPU time	GPU time (msec)				speedup
	(msec)	upload	Process	download	total	
128 x 128	6.151	0.073	0.187	0.101	0.360	17.086
256 x 256	21.680	0.229	0.285	0.279	0.793	27.339
512 x 512	89.909	0.876	0.718	1.032	2.627	34.225
1024 x 1024	442.490	3.131	2.429	3.078	8.639	51.220
2048 x 2048	1812.672	12.197	9.266	10.421	31.884	56.852
4096 x 4096	7088.614	48.353	36.570	40.288	125.211	56.613

TABLE III
INVERSE WAVELET TRANSFORM EXECTION TIME

Size	CPU time (msec)	GPU time (msec)				Speedup
		Upload	Process	Download	Total	
128 × 128	6.274	0.208	0.194	0.087	0.489	12.830
256 × 256	21.946	0.282	0.310	0.285	0.877	25.024
512 × 512	90.931	0.893	0.768	1.055	2.717	33.467
1024 × 1024	447.009	3.204	2.626	2.970	8.799	50.802
2048 × 2048	1833.699	12.312	10.046	10.441	32.798	55.909
4096 × 4096	7195.311	48.409	39.670	40.452	128.531	55.981

GPU execution time is found by adding the uploading and downloading time of the image file to the GPU by the CPU and the processing time of the image file by the GPU. The speedup columns of TABLE II and TABLE III indicate the ratio of the CPU execution time to the GPU execution time. This ratio clearly indicates that G80 is always faster than the CPU. For large size images the speedup ratio increases and reaches its maximum of about 65.

IV. CONCLUSION

Image compression based wavelet transform consumes its time mostly in the wavelet transformation. Using GPU to perform transformation now relieves the CPU to perform other tasks. This result opens up other possibilities to use GPUs for more advanced application.

REFERENCES

[1] Topiwala, Pankaj N., Wavelet Image and Video Compression, Kluwer Aca-demic Publishers.
[2] Miano,John, Compressed Image File Formats JPEG, PNG, GIF, XBM, BMP, Addison Wesley Longman, Inc, 1999.
[3] Tokdemir, Serpil, Digital Compression on GPU, Georgia State University,2006
[4] Podlozhnyuk, Victor, Image Convolution with CUDA, NVIDIA Corporation, 2007.
[5] Klawonn, Frank, Introduction to Computer Graphics Using Java 2D and 3D, Springer-Verlag London Limited, 2008. http://dx.doi.org/10.1007/978-1-84628-848-7
[6] Misiti, Michel; Misiti, Yves; Oppenheim, Georges; and Poggi, Jean-Michel, Wavelets and their Applications, ISTE Ltd, 2007. http://dx.doi.org/10.1002/9780470612491
[7] Salomon, David, Data Compression The Complete Reference, Third Edition, Springer-Verlag New York Inc, 2004.
[8] Koschan, Andreas; Abidi, Mongi, Digital Color Image Processing, John Wi-ley & Sons, Inc, 2008
[9] Getreuer, Pascal, Filter Coe_cients to Popular Wavelets , May 2006.
[10] NVIDIA CUDA Compute Uni_ed Device Architecture Programming Guide, Version 1.1, NVIDIA Corporation, 2007.