© 2018 IJSRSET | Volume 4 | Issue 4 | Print ISSN: 2395-1990 | Online ISSN : 2394-4099
Themed Section : Engineering and Technology

Efficient parallelization of Inverse DWT using GPGPU

Shailja Maniya[1], Bakul Panchal[2]

[1]Research Scholar, M.E. (I.T.), I.T. Department, L.D. College of Engineering Gujarat Technological University, Ahmedabad, Gujarat, India

[2]Assistant Professor, M.E. (I.T.), I.T. Department, L.D. College of Engineering Gujarat Technological University, Ahmedabad, Gujarat, India

ABSTRACT

Satellite images are gaining more and more popularity in our daily life as they are helpful during situations like natural calamities or warfare. In order to save bandwidth as well as to speed up data transfer, compression can be used to download satellite images on the earth. The Consultative Committee for Space Data Systems (CCSDS) had proposed an image data compression standard (CCSDS-IDC) for satellite image compression. This standard provides good compression performance using Discrete Wavelet Transform (DWT) and Bit Plane Encoder. As Discrete Wavelet Transform (DWT) is time consuming, to meet real time requirement this data should be decompressed as soon as massive stream of bits downlinked on the earth. In this research work, efficient GPGPU based IDWT (Inverse DWT) computation gives better time efficiency than CPU implementation.

Keywords: GPGPU, CCSDS, Discrete Wavelet Transform (DWT), CUDA, NVIDIA.

I. INTRODUCTION

The CCSDS Image Data Compression is the most widely used particularly for the grayscale image data. This algorithm is very useful for any imaging instruments application. The algorithm is designed in such a way that its complexity remains sufficiently low so that it can be feasibly implemented on high-speed hardware.

This algorithm is intended to be used for on-board spacecraft. Compression is used in order to save bandwidth usage & storage space required to save the image data as well as time. On the other side, real time need is to decompress the two dimensional image data as soon as it is downlinked on the earth.

Various data Compression technique uses wavelet transformed data for better compression performance. There are basically two types of compression supported by this CCSDS standard i.e. Lossless Data Compression and Lossy Data Compression. In Lossless Data Compression, data is compressed in such a way that it can be recovered easily on decompression whereas in Lossy data compression technique, data cannot be reproduced without some distortion. Lossy Compression technique uses 9/7 Float Discrete Wavelet Transform; Lossless Compression Technique uses 9/7 Integer Discrete Wavelet Transform. As Integer 9/7 Discrete Wavelet Transform is the most widely used algorithm for the compression as well as time consuming one, it can be efficiently parallelized using GPU to enhance the performance of the compression system.

II. LITERATURE SURVEY

Changhe Song, Yunsong Li, and Bormin Huang at el. [2] has implemented the decoding system for satellite images. They have implemented a wavelet based decoding system which contains SPIHT with Reed-Solomon decoding. In this paper, they have used float 9/7 inverse DWT as a wavelet transform with SPIHT and Reed-Solomon decoding. They are getting IDWT as the most time consuming process of the decoding system. They have used General Purpose Graphics Processing Unit (GPGPU) for the faster computation of IDWT. They have also used shared memory for the better time performance of the IDWT computation.

Abhishek S. Shetty, Abhijit V. Chitre and Yogesh H. Dandawate at el [3] has done the comparison for the wavelet and Inverse wavelet transform on different platforms. They have used 9/7 Integer Discrete wavelet transform and its inverse for this comparison. Three different platforms are used which are MATLAB, Python using OpenCV and Python using PIL (Python Imaging Library). As a result, the comparison chart is prepared as shown in fig. 9. It is observed that Pthon-OpenCV gives better result in terms of time and it is also open source. These results are being obtained using image size starting from 512x512 to 6000x6000.

Anastasis Keliris, Vasilis Dimitsasy, Olympia Kremmyday, Dimitris Gizopoulosy and Michail Maniatakosz at el [4] has implemented the Discrete wavelet transform on multi core cpu, Many Integrated Cores (MIC) cpu as well as on NVIDIA GPU. In this paper, 9/7 integer DWT is used as a wavelet transform. They have used the two evaluation systems; one is using AMD Pheom II X4 965 with Nvidia Tesla C2070 & another one is Intel Xeon E5-2680 with Intel Xeon Phi 5110P. They have used the algorithm for the transpose of the matrix because this operation is memory intensive operation.

Christofer Schwartz, Marcelo S. pinho, at el [9] has implemented CCSDS-122 DWT based compressor on the CPU as well as on the GPU. CPU specifications are: Intel Core i7 - 3610QM (third generation) with 4 cores (8 threads) working at a frequency of 2; 241; 003 KHz and GPU specifications are NVIDIA GeForce GT 630M (2.1 Streaming Multiprocessor Capability) — this GPU has two multiprocessor (MP) with 48 cores each (total of 96 cores) working at a clock of 950; 000 KHz. In this paper, they have implemented bit-plane encoder on the Host side (CPU) and DWT is performed on device (GPU). Timing given by the host and host + device system is analysed in this paper. Energy consumption of the GPU + CPU system was less than CPU system.

III. INVERSE DISCRETE WAVELET TRANSFORM

The Inverse Discrete Wavelet Transform (IDWT) can be considered as correlation module, as with every level of transform data gets correlated. Inverse Integer 9/7 DWT maps two sets of wavelet coefficients, a low-pass set, *Cj,* and a high-pass set, *Dj,* back to a signal vector *xj*. Special boundary filters are required at either end of the data sets, for *j*=0, *j*=1, *j*=2*N*-3, and *j*=2*N*-1.

$$x_1 = D_0 + \lfloor \frac{9}{16}(x_0+x_2) - \frac{1}{16}(x_2+x_4) + \frac{1}{2} \quad (1)$$

$$x_{2j+1} = D_j + \lfloor \frac{9}{16}(x_{2j}+x_{2j+2}) - \frac{1}{16}(x_{2j-2}+x_{2j+4}) + \frac{1}{2} \rfloor$$

for j=1,…,N-3 (2)

$$x_{2N-3} = D_{N-2} + \lfloor \frac{9}{16}(x_{2N-4}+x_{2N-2}) - \frac{1}{16}(x_{2N-6}+x_{2N-2}) + \frac{1}{2} \rfloor \quad (3)$$

$$x_{2N-1} = D_{N-1} + \lfloor \frac{9}{8} x_{2N-2} - \frac{1}{8} x_{2N-4} + \frac{1}{2} \rfloor \quad (4)$$

$$x_0 = C_0 + \lfloor -\frac{D0}{2} + \frac{1}{2} \rfloor \quad (5)$$

$$x_{2j} = C_j + \lfloor -\frac{D_{j-1}+Dj}{4} + \frac{1}{2} \rfloor \text{ for } j=1,\ldots, N-1 \quad (6)$$

IV. IMPLEMENTATION DETAILS

This inverse DWT problem can be parallelized. As we are going to use three level inverse DWT and each level computation depends on the previous level result, it cannot be parallelized. The parallelizable task is to perform the computation of inverse horizontal filter and inverse vertical filter.

Eq. (5) & (6) should be computed in parallel for inverse horizontal & inverse Vertical filter for each pixel values LL, LH as well as HL, HH subbands. After that, Eq. (1) to (4) can be computed in parallel for inverse horizontal & inverse Vertical filter. Total number of threads launched for each filter is same as total pixel values of the subband. One thread computes one-pixel value for next level subband.

From the given equations we can say that odd elements can only be computed after the computation of even elements is completed. Eq. (1) to (4) depends on Eq. (5) & (6). In our implementation, four kernels are made to compute one level inverse 2-D DWT. The fetch and copy step of this inverse vertical filter and inverse horizontal filter is shown in below figutr 2

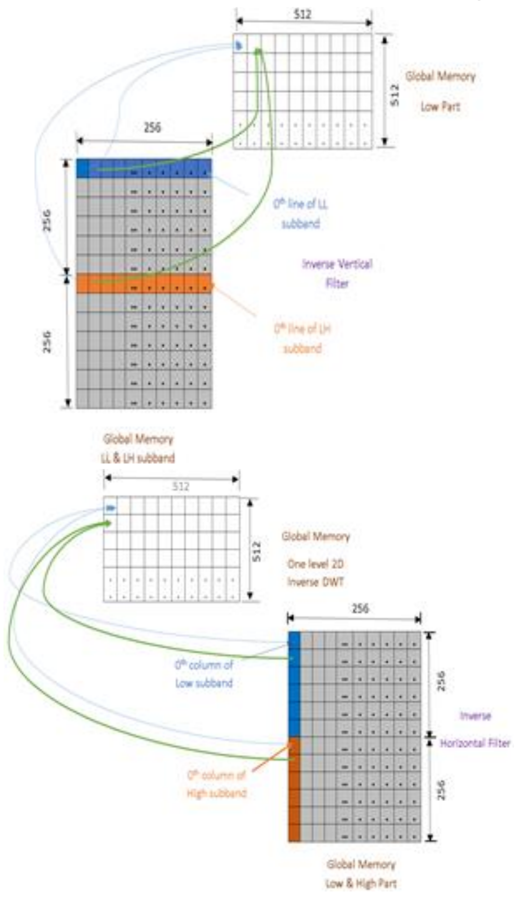

Figure 2. Fetch and copy step for inverse vertical filter and inverse horizontal filter

In the GPU computation, the memory access pattern matters a lot. There are many types of memory defined in CUDA programming model. One should choose the memory access pattern very carefully for achieving desired speedups.

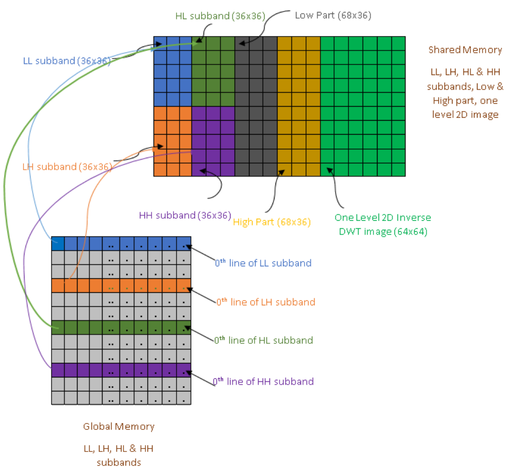

Figure 3. Memory load step of kernel in shared memory

- Inverse Discrete Wavelet Transform Optimization

In the parallel implementation, the time consuming part is the memory load store operation. Considering previous version of decompressor, result of all four kernels are stored in the global memory. Thus, in one level inverse 2-D DWT is accessing global memory six times. AS Time required for memory load store operation in the global memory has significant impact on the performance, access global memory for one time during all computation of one level 2-D Inverse DWT to optimize the performance. It can be done by using tiling approach and shared memory.

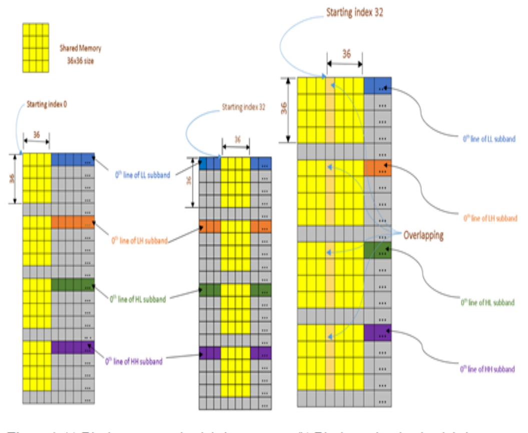

Figure 4. (a) Block movement in global memory (b) Block overlapping in global memory

In this optimized version, we have done the computation on block size of 36x36 by launching only 32x32 threads in the one block. Here we have hallo region of 2 columns and 2 rows. We require total 36x36 load operation for computation of 32x32 block size. We also have overlapping in loading in tiling approach. This Loading operation from global to shared memory is shown in figure 3.

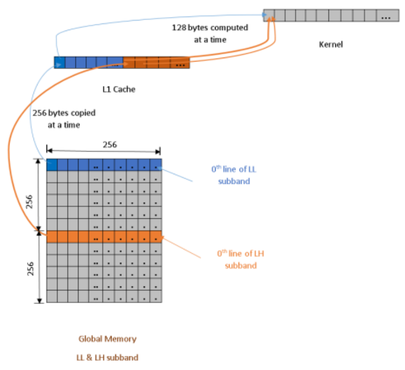

Figure 5. Fetch & Computation step for Eq. (6) for Inverse Vertical Filter

In the implementation, we have used Tesla K40c Accelerator for the parallel computation of Inverse Discrete wavelet transform. In this GPU, when we read any data from the global memory it takes 256 bytes (128 pixel values) at a time. When we request again same values then it is fetched from the L1 cache which reads 128 bytes (64 pixel values) at a time. In previous case, we were using only 32 pixel values out of 64 pixel values at a time for the computation of IDWT.

In the optimized version, one thread is going to compute two pixel values at a time which in turns uses all 64 pixel values which are being fetched in one clock cycle from L1 cache. Thus, we get higher hit ratio for L1 cache and speedup is achieved. This memory fetch and computation step is as shown in figure 5.

V. EXPERIMENTAL RESULTS

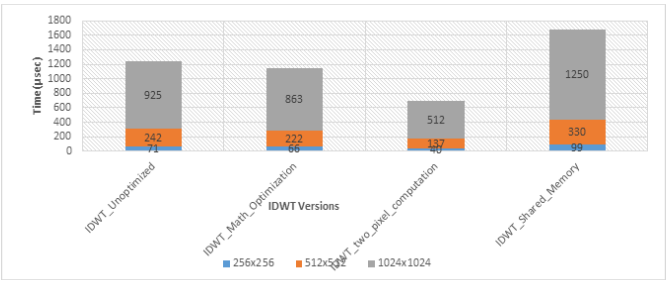

Figure 7. Time Performance Comparison chart for Decompressor Versions

The computation time for GPU depends upon memory load store operations as well as kernel computation time. IDWT_two_pixel_computation takes least amount of time for kernel computation whereas IDWT_Shared_Memory takes least number of load-store operations. The comparison results are as shown in the below graph.

VI. CONCLUSION

The inverse DWT problem can be computed in parallel in order to get decompression of satellite image as soon as it is downlinked on the earth. For the parallel implementation, GPU is very useful. NVIDIA CUDA programming is used to program GPU for the parallel implementation. From all the versions of IDWT, the Two_Pixel_Computation gave best results in terms of time. It is also analysed that how we are going to use parallelism in our implementation as well as the memory access pattern has significant impact on the performance. IDWT_two_pixel_computation gives best result in terms of time among all implementations of IDWT considering time for memory load-store operation as well as for kernel computation.

VII. REFERENCES

[1]. "Image Data Compression", Recommendation for space data system standards, CCSDS 122.0-B-1. Blue Book, November 2005.

[2]. Changhe Song, Yunsong Li, and Bormin Huang, "A GPU-Accelerated Wavelet Decompression System with SPIHT and Reed-Solomon Decoding for Satellite Images", IEEE journal of selected topics in applied earth observations and remote sensing, vol. 4, no. 3, september 2011.

[3]. Abhishek S. Shetty, Abhijit V. Chitre and Yogesh H. Dandawate, "Time Efficiency Comparison of Wavelet and Inverse Wavelet Transform on Different Platforms", International Conference on Computing Communication Control and automation (ICCUBEA) IEEE-2016.

[4]. Anastasis Keliris, Vasilis Dimitsasy, Olympia Kremmyday, Dimitris Gizopoulosy and Michail Maniatakosz, "Efficient parallelization of the Discrete Wavelet Transform algorithm using memory-oblivious optimizations", 25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), pp.25-32, 2015

[5]. John Nickolls, "GPU Parallel Computing Architecture and CUDA Programming Model", Hot chips 19 Symposium(HCS) IEEE, pp.1-12, 2007

[6]. Khoirudin and Jiang Shun-Liang, "Gpu application in cuda memory", Advanced Computing: An International Journal (ACIJ), Vol.6, No.2, pp.1-10, March 2015

[7]. NVIDIA, "NVIDIA’s Next Generation CUDATM Compute Architecture: Kepler TM GK110/210", United States, 2014.

[8]. NVIDIA, "Cuda C Programming Guide", United States, September 2017.

[9]. Christofer Schwartz, Marcelo S. pinho,"an energy consumption analysis of ccsds image compressor running in two different platforms", IGARSS-IEEE, pp. 1640- 1650, 2014.