International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 5 Issue: 6 1140 – 1145

Discrete Wavelet Transformation Implementation in GPU through Register Based Strategy

Hemkant Balasaheb Gangurde
Department of Computer Engineering
MET's Institute of Engineering
Nashik, India
gangurdehemkant@gmail.com

M. U. Kharat
Department of Computer Engineering
MET's Institute of Engineering
Nashik, India
mukharat@rediffmail.com

***Abstract*—** The significant architectural changes made by Nvidia during the launch of Kepler architecture in 2012, upgraded its GPUs with greater register memory and rich instructions set to have communication between registers through available threads. This created a potential for new programming approach which uses registers for sharing and reusing of data in the context of the shared memory. This kind of approach can considerably improve the performance of applications which reuses implied data heavily. This work is based upon of register-based implementation of the Discrete Wavelet Transform (DWT) with the help of CUDA and openCV. DWT is the data decorrelation approach in the area of video and image coding. Results of this particular approach indicate that this technique performs at least four times better than the best GPU implementation of the DWT in past. Experimental tests also prove that this approach shows the performance close to the GPUs performance limits.

Keywords- *CUDA, DWT, Kepler, Nvidia, OpenCV*

*****

I. INTRODUCTION

The computational power of GPUs is growing notably day by day. Previously GPUs were only used to decrement the graphics rendering burden due to CAD (computer-aided design) or high graphics video games on CPUs. GPUs are currently used for mainstream applications as well. During the evolution of GPUs, it has gone through major changes in its architecture. The most important change was the release of Compute Unified Device Architecture (CUDA) in 2006 of the Nvidia, which provided architectural tools for general purpose computing together along with C-compiler for the GPU to have GPU programming. While using GPU for the implementation of mainstream application one must have to take care so that the potential of the GPU capacities get fully exploited. Managing the data internally is the important aspect. The important factor in GPU-based implementation is storing required data in the legitimate memory areas. Memory space of GPU is divided into three areas: global memory, shared memory, and register-based memory. Global memory is the largest, situated off-chip DRAM which shows the largest latency. Register and the shared memory are present on-chip and do get managed accordingly. In comparison, they are much faster in comparison with global memory, but their size is much smaller. The important deviation between the register memory and the shared memory is that use of shared memory is more common in order to store and reusing intermediate results and sharing data between threads efficiently. The arithmetic and logical operations are performed in register memory where threads are private in registers.

During the launch of CUDA, Nvidia has released the guidelines [2] which have strongly recommended using shared memory space for the operations such as sharing and reusing of data. These recommendations were challenged by Volkov and Demmel [3][4], who explored that an extreme use of the shared memory may decrement the level of performance. Three factors are responsible for this. The first one is the bandwidth associated with shared memory that, though it is very high, it might act as the bottleneck for the applications which uses previously used data heavily. The next factor is that ALU dependent operations whose data is present in the shared memory space must be required to move it to registers before performing the operations. The last one is that the size of shared memory space is considerably lesser in comparison with register memory space. Volkov and Demmel [3][4] shown that it is possible to gain the GPU's level of performance by directly using the register memory, by decreasing the interference of the shared memory. These results suggested that performance can be maximized by making register memory space as local storage place when data reusing is the must.

In Kepler architecture launched in 2012, the size of the register memory space is made twice as previous, and the quantity of registers which single thread can able to manage is made four times as that of the previous, also a new instructions set is introduced in order to enable the data sharing in the register space. These improvements helped register based implementations in order to code the GPU. Which is an emerging programming approach to have the better and faster implementation of the applications which can able to use GPUs efficiently.

The DWT (Discrete Wavelet Transformation) is a wavelet transformation implementation based on a discrete set of the wavelets and translations following some predefined rules. Discrete Wavelet Transformation decomposes the signal into mutually orthogonal wavelets set, this is an important deviation from the continuous wavelet transform or its implementation for the discrete time series sometimes called discrete-time continuous wavelet transform.

A GPU-based implementation using Nvidia hardware needs the understanding of CUDA. CUDA stands for Compute Unified Device Architecture. CUDA is launched by Nvidia in 2006, as a platform for parallel computation and an API for GPU programming. CUDA permitted software and hardware developers to make use of CUDA-enabled graphics processing unit for general purpose processing which is also known as General Purpose computing on Graphics Processing Units (GPGPU). It creates a software level access which allows direct access to the GPU's instruction and parallel computational elements, in order to execute compute kernels. This platform can be used with high-level programming languages such as C, C++, and FORTRAN. This enables to specialists in this languages to program the GPU which was only possible previously for the experts in technologies such as OpenGL and Direct3D. Furthermore, CUDA is compatible with frameworks like OpenACC as well as OpenCL.

This work has included OpenCV to perform experiments on images in order to obtain the intensity of the samples (pixels). OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV is mainly written in c++ which is the container of more than 2000 useful algorithms for the image processing and machine learning. Its interfaces are available for other platforms like Java and .Net. The purpose of inclusion of openCV in this work was to perform operations on real images in order to provide the results for the register based implementation of DWT.

This work explores the register based implementation strategy for the DWT [1] with the help of openCV. Discrete Wavelet Transformation which is a technique for data decorrelation in the area of video and image programming. It's usage found in international standards of compression, which include JPEG, also in a number of coding schemes which includes SPIHT, EBCOT, or SPECK. In order to implement the DWT using GPU as hardware data reuse must be considered significantly. Various strategies are present in the literature so that data could be reused efficiently. The use of a register based implementation allows an approach which is very much deviated from previous methods. Register based implementation of DWT needs to be implemented from start. The significant aspects of this strategy are partitioning of data and mapping of thread and associated data. This strategy gains speedups of 4 times in comparison with the best techniques present in the previous work.

II. LITERATURE SURVEY

Since a lot of work and efforts has been taken while providing the efficient way to implement the DWT. In this section, we had discussed existing techniques used to implement DWT.

The implementations before emerging of CUDA, DWT were employed over a number of devices and different programming languages based upon GPU as a hardware. The implementation proposed in [5][6], was based on OpenGL which introduced a decomposition of wavelets and recreation of the algorithm, which directly deals with the graphics hardware of OpenGL workstations as well as increases the speed of time taking filtration steps resulting into time-saving. This particular approach has used the convolution method as well as color matrix combined with OpenGL's in order to scale images in the process of copying the instructions, they performed all required stages of 2D tensor product wavelet filtering instead of copying data from or to the machine's main memory, resulting into avoiding typical bottlenecks which may occur in the visualization cycle. OpenGL is a cross language and platform API for rendering two-dimensional and three-dimensional graphics. This application programming interface is majorly used to have communication with GPU in order to gain fast graphics rendering. Whereas [7], [8] employed OpenGL as well as Computer graphics together. Lots of these older approaches were based on convolution technique.

DWT was evaluated for the first time using lifting scheme [8]. In this work they explored a simple and powerful and efficient solution in order to implement two-dimensional DWT on the consumer level hardware (GPU). This approach does not need any expensive hardware to achieve better performance. This method proved that implementation can be done on any SIMD graphics processing unit. This technique unites the mathematically

 ISSN: 2321-8169
1140 – 1145

deviated forward and inverse discrete wavelet transformations. Even the convolution-based method was popular previously as the lifting scheme needed intermediate results to be shared in between the coefficients. Tools were unavailable to implement lifting scheme efficiently in the past. Which was confirmed in [9] experimentally, in [9] the two approaches, convolution and the lifting scheme were implemented and compared for the performance together.

The general purpose architecture of graphics processing unit (GPU), as well as associated tools for programming GPU, were lacking in the previously mentioned pre-CUDA implementations. The tasks involved in the DWT must be related to graphics operations, which found to be limited in past. Even these techniques were far faster compared to a CPU-based implementation, their performance could have been more with current GPUs which having an advanced memory arrangement for GPU programming. One of the most important thing in present CUDA based implementations is the way in which image partitioning takes place in order to allow parallel implementation scheme. There are three main strategies applied for partitioning, called row-column scheme, row-block scheme, and block-based scheme.

The earliest implementation based upon CUDA of DWT was mentioned in [10]. Which uses the row-column approach. At a start, a block of the thread does load a row of an image in shared memory space then threads compute horizontal filtering process on that row. First CUDA based implementation which combines lifting scheme as well as block-based scheme to implement DWT was addressed in [11]. The important factor of this approach was that it decreases the number of transfers to the global memory as it evaluates horizontal and the vertical filtering in a one computation step. In this method, the image is partitioned into rectangular blocks that need to be loaded in the shared memory by a thread block. Then horizontal filtering and the vertical filtering are performed to these blocks, in this, there is no need of memory transfer operation or need of performing transpose of the matrix. The setback in this kind of approach is that there are dependencies of data between blocks which are next to each other. These dependencies are not taken care in [11]. The simple solution here is to extend all blocks with a certain number of rows and columns which overlap with blocks next to each other. The fastest implementation of the Discrete Wavelet Transformation was present in the literature, which was explored in [12]. In this approach, the row-block partitioning technique was used, which can work on any number of dimensions. They compared their method to an optimized CPU implementation of the lifting scheme, to another (non-CUDA based) GPU wavelet lifting method, and also to an implementation of the wavelet transform in CUDA via convolution.

III. SYSTEM ARCHITERCTURE

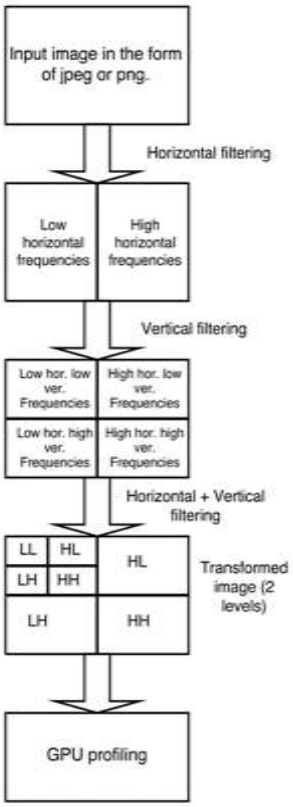

Figure 1. DWT two level decomposition architecture

Figure 1. shows the basic system architecture. Input is provided as an image to the system and based upon the number of decomposition levels applied the operations will be carried out.

In Figure 1. the two decomposition levels has been shown which may vary based upon the requirements. Finally after performing the DWT operation profiling of GPU will be executed in order to express various GPU parameter (e.g. execution time, throughput).

IV. ALGORITHM

The algorithm provided below gives a glance at CUDA kernel implementation in the register based strategy. This is an algorithm for the forward Discrete Wavelet Transformation. This CUDA kernel function is executed by all the threads present in the each warp simultaneously. This algorithm takes parameters such as (i) thread identifier *TI* (ii) first column and row of an image related to the block computed by current warp (i.e., *P; Q*) (iii) starting column and row of the wavelet sub-bands in which the current warp must leave the resulting coefficients (i.e., *PS*; *QS*). The height of the block is denoted by *B'* and is a constant.

 ISSN: 2321-8169
1140 – 1145

1: Allocation of *K[B'][2]* in register based memory space
2: *y = 0*
3: *while y < B'*
4: *start of while loop*
5: *K[y][0] = GM[Q+y][P+TI*2]*
6: *K[y][0] = GM[Q+y][P+TI*2+1]*
7: *y = y+1*
8: *end of while loop*
9: *j= 0*
10: while *j < J'*
11: start of while loop
12: *y=0*
13: *while y < B'*
14: start of while loop
15: $K' = \Phi(K[y][0],TI+1)$
16: $K[y][1] = K[y][1]- \alpha^{j}(K[y][0]+K')$
17: $K' = \Phi(K[y][1],TI-1)$
18: $K[y][0] = K[y][1]- \beta^{j}(K[y][1]+R')$
19: *y = y+1*
20: end of while loop
21: *j = j+1*
22: end of while loop
23: *j=0*
24: *while j < J*
25: *start of while loop*
26: *y =1*
27: *while y < B'*
28: *start of while loop*
29: $K[y][0]= K[y][0] - \alpha^{j}(K[y-1][0] +K[y+1][0])$
30: $K[y][1]= K[y][1] - \alpha^{j}(K[y-1][1] +K[y+1][0])$
31: *y = y+2*
32: *end of while loop*
33: *y=0*
34: *while y < B'-1*
35: *start of while loop*
36: $K[y][0]= K[y][0] - \beta^{j}(K[y-1][0] +K[y+1][0])$
37: $K[y][1]= K[y][1] - \beta^{j}(K[y-1][1] +K[y+1][0])$
38: *y = y+2*
39: *end of while loop*
40: *j = j+1*
41: *end of while loop*
42: *for* $y \in \{2J,2J+2,.....Y' -2J\}$ do
43: $GM[Q_{LL} + y/2][P_{LL}+TI] = K[y][0]$
44: $GM[Q_{HL} + y/2][P_{HL}+TI] = K[y][1]$
45: *end for*
46: *for* $y \in \{2J+1,2J+3,.....Y' -2J+1\}$ do
47: $GM[Q_{LH} + y/2][P_{LH}+TI] = K[y][0]$
48: $GM[Q_{HH} + y/2][P_{HH}+TI] = K[y][1]$
49: *end for*

In first stage, the algorithm given below do reserve the registers required to the thread. Here registers are denoted by *K* and global memory by *GM*. From line 2 to 8, thread are reading from the global memory. Operation of horizontal filtering is from line 9 to 22 and vertical filtering is from 23 to 41. Line 42 to 49 results are saved in global memory.

V. MATHEMATICAL MODELING

Let S be a system.

S= {I, F, O} Where,
I represent the set of inputs:
I= {I1, I2, I3, I4, I5, I6} where,
I1 ⟶Which filter to be used CDF 5/3 or CDF 9/7 will be provided as a binary input.
I2 ⟶Number of random generated images.
I3 ⟶Size in both dimensions of the random generated input image employed.
I4 ⟶Length of the Data block computed by each warp.
I5⟶Whether to use shuffle instructions or intermediate buffer.
I6⟶ Number of DWT decomposition levels applied.

F is the set of functions:
F= {F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12}; where
F1⟶ Performing 5/3 CDF.(FORWARD).
F2⟶ Performing 9/7 CDF.(FORWARD).
F3⟶ Performing 5/3 CDF.(REVERSE).
F4⟶ Performing 9/7 CDF.(REVERSE).
F5⟶ Performing vertical filter 5/3 (FORWARD).
F6⟶ Performing vertical filter 5/3 (REVERSE).
F7⟶ Performing vertical filter 9/7 (FORWARD).
F8⟶ Performing vertical filter 9/7 (REVERSE).
F9⟶ Performing Horizontal filter 5/3 (FORWARD).
F10⟶ Performing Horizontal filter 5/3 (REVERSE).
F11⟶ Performing Horizontal filter 9/7 (FORWARD).
F12 ⟶Performing Horizontal filter 9/7(REVERSE).

O is the set of outputs:
O= {C}
C⟶ The comparison of various GPU parameters of forward and reverse DWT.

VI. ANALYSIS OF RESULTS

The experimental results mentioned in this section are carried out with a Nvidia GTX 650M CUDA v8.0 compiler and on the operating system having a block size 512 bytes. This GPU has 15 SMs and a peak global memory bandwidth of 336 GB/s. These results are for the block size 128, which is one of the recommended sizes to generate maximum performance. Results here are collected with the help of Nvidia profiler tool. The result indicated here are for both

 ISSN: 2321-8169
1140 – 1145

5/3 CDF and 9/7 CDF with the help of image of size 768*768 and for 2160* 2160.

The first experiment evaluates the performance achieved by the proposed method and compares it to the best implementation in the literature based upon the shared memory strategy [12]. The strategy used in [12] is to obtain the maximum performance using shared memory usage. Figure 2. depicts the result for 5/3 CDF filter bank using the image size of 768*768. As per the results, it gives a significant amount of execution time difference (up to four times) between the shared memory based approach and the proposed (register based approach). In Figure.2 results are for the two levels of DWT decomposition. Decomposition levels can be changed as per the requirement of the application.

The size of an image is directly proportional to the execution time of DWT kernels. Figure 3. explores the same operation i.e. DWT using 5/3 CDF using a large image of size 2160*2160.

Figure 2. and Figure 3. results are for 5/3 CDF filter bank. 9/7 CDF is another popular filter bank which is used in jpeg standard. Figure 4. provides are the results for 768*768 image DWT using 9/7 CDF filter bank. When register based strategy is used shuffled instructions are enabled. It is easy to conclude from Figure 2. and Figure 4. that the 9/7 CDF filter bank takes more time compared to 5/3 CDF for the same samples of input.

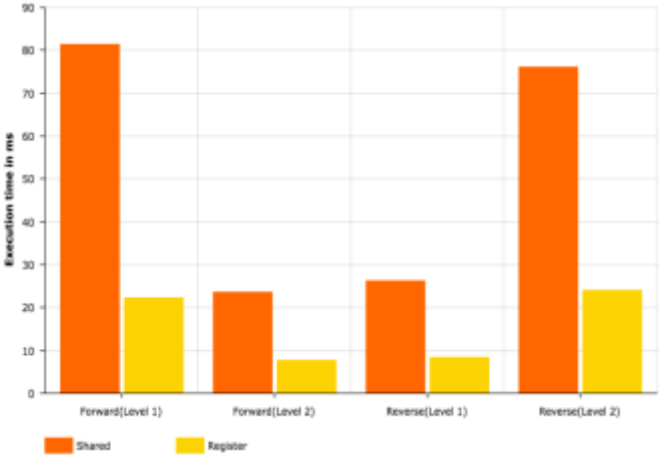

Figure 2. The results comparison for the cuda DWT kernels for an image of size 768*768 with 5/3 CDF

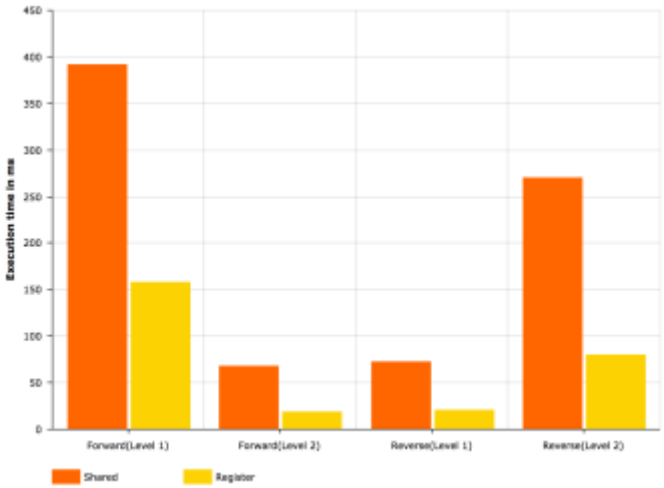

Figure 3. The results comparison for the cuda DWT kernels for an image of size 2160*2160 with 5/3 CDF

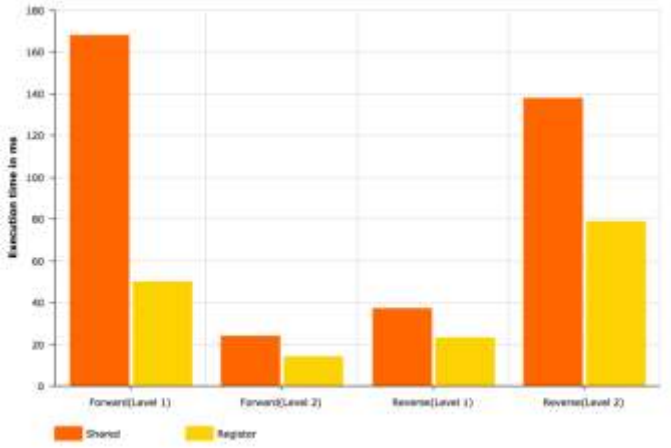

Figure 4. The results comparison for the cuda DWT kernels for an image of size 768*768 with 9/7 CDF

The Figure. 5 explores the result of DWT using shared memory and register based strategy for image size 2160*2160 using 9/7 CDF.

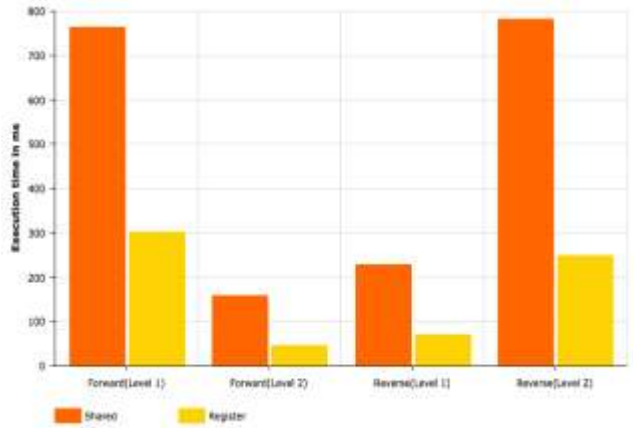

Figure 5. The results comparison for the cuda DWT kernels for an image of size 2160*2160 with 9/7 CDF

VII. CONCLUSION

This work provides the implementation of DWT (Discrete Wavelet Transform) in GPU using register-based strategy with the help of lifting scheme. Register based strategy for DWT implementation does become possible in recent CUDA based architecture because of increase in register-based memory size and the introduction of the instructions for the data sharing such as shuffle instructions. Register based implementation in this system makes use of openCV an open-source technology for image processing to extract the image samples. It is possible that future generations of GPU may be changed in the context of architecture, but it is likely that these future GPUs still provide or increase the amount of register memory. In this strategy, all the operations are carried out using register-based memory which eliminates the requirement of data movement to conduct the ALU based operations which were acting as performance bottleneck in case of the shared memory based implementation. This system uses an optimized block based partitioning scheme. It has a process of thread to data mapping which allows assigning of the warps in order to compute entire data of the single block. Experimental results do indicate the significant decrease in the execution times of DWT kernels due to the usage of register memory space. Evidence in results indicates that register based implementation of DWT provides performance gain up to four times compared with the best implementation found in the literature. The execution time of DWT kernels is directly proportional to the size of an image. This system does provide the implementation for both 9/7 CDF and 5/3 CDF filter banks which are standard in JPEG 2000 for the progressive lossy and lossless compression techniques respectively. Experiments do conclude that 9/7 CDF is more expensive in the context of kernel execution timings as compared to 5/3 CDF filter bank. Experimental analysis proves that this strategy provides a memory-bounded implementation. The bandwidth of global memory is near to the maximum which can be achieved. As almost all global memory traffic can't be avoided, it can be concluded that execution times achieved for the DWT kernels are near to the hardware limitations of current architecture.

References

[1] Pablo Enfedaque, Francesc Aul-Llinas and Juan C. Moure, “Implementation of the DWT in a GPU through a Register-based Strategy”, in IEEE Trans.Parallel Distrib. Syst, 2015, pp 3394-3406.

[2] Nvidia, CUDA C Programming guide, 2014. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide.

[3] V. Volkov and J. W. Demmel, ``Benchmarking GPU’s to tune dense linear algebra", in Proc. ACM/IEEE Conf. Supercomputing, Nov. 2008, pp. 31-42.

[4] V. Volkov, ``Better Performance at Lower Occupancy", in IEEE Int. Conf. Image Pro., 2010.

[5] F. N. Iandola, D. Sheffield, M. Anderson, P. M. Phothilimthana, and K. Keutzer, ``Communication-minimizing 2D convolution in GPU registers", in Proc. IEEE Int. Conf. Image Process., Sep. 2013, pp. 2116-2120.

[6] M. Hopf and T. Ertl, ``Hardware accelerated wavelet transformations", in Proc. EG/IEEE TCVG Symp, 2000, pp. 93-103.

[7] A. Garcia and H.-W. Shen, ``GPU-based 3D wavelet reconstruction with tileboarding", in Vis. Comput, 2005, pp. 755-763.

[8] T. T. Wong, C.-S. Leung, P.-A. Heng, and J. Wang, ``Discrete wavelet transform on consumer-level graphics hardware", in IEEE Trans. Multimedia, 2007, pp. 668-673.

[9] C. Tenllado, J. Setoain, M. Prieto, L. Pinuel, and F. Tirado, ``Parallel implementation of the 2D discrete wavelet transform on graphics processing units: filter bank versus lifting", in IEEE Trans. Parallel Distrib. Syst, vol. 19, no. 3, 2008, pp. 299-310.

[10] J. Franco, G. Bernabe, J. Fernandez, and M. E. Acacio, ``A parallel implementation of the 2D wavelet transform using CUDA", in Proc. 17th Euromicro Int. Conf. Parallel, 2007, pp. 111-118.

[11] J. Matela, ``GPU-based DWT acceleration for JPEG2000", in Proc. Annu. Doctoral Workshop Math. Eng. Meth. Comput. Sci, 2009, pp. 136-143.

[12] W. J. van der Laan, A. C. Jalba, and J. B. Roerdink, "Accelerating wavelet lifting on graphics hardware using CUDA", in IEEE Trans. Parallel Distrib, 2011, pp. 132-146.