Block-based Approach to 2-D Wavelet Transform on GPUs

Michal Kula, David Barina, and Pavel Zemcik

Brno University of Technology
Brno, Czech Republic
{ikula,ibarina,zemcik}@fit.vutbr.cz

Abstract. This paper introduces a new approach to computation of 2-D discrete wavelet transform on modern GPUs. The proposed approach involves block-based processing enabling one seamless transform even for high resolution input data. Inside the blocks, two distinct methods can be used – either separable or non-separable 2-D lifting scheme. Furthermore, the paper presents a comparison of the proposed approach under different conditions to the best existing methods, whereas our approach consistently outperforms the other ones. Our methods are implemented using the OpenCL framework and tested on a wide range GPUs.

Keywords: discrete wavelet transform, image processing, lifting scheme, graphics processing units, memory barrier

1 Introduction

The 2-D discrete wavelet transform (DWT) is the signal-processing transform suitable for decomposition of the analysed 2-D signal into several scales. On each scale, three directional subbands are formed. These are usually referred to as HL, LH, and HH subbands. The 2-D transform is defined as separable product of 1-D transforms performed sequentially on rows and columns (or vice versa). Each of these one-dimensional transforms can be computed through either the convolution or the lifting scheme. Different strategies of 2-D DWT implementation were developed for various computational platforms.

In this paper, we focus on implementation of DWT using modern graphics cards (GPU) capable of a general-purpose computing. In these architectures, the GPU contains thousands of stream processors that are clustered into blocks. All processors in each block execute the same instruction with different operands at one time. The blocks are grouped into multiprocessors which form the basic functional units of the GPUs. The thread scheduler allocates as many work groups to multiprocessors as their resources allow. The work groups are defined as a group of threads that can interoperate with each other using the local memory and memory barriers. Thus, the resources, such as the local memory size, should be minimized. The allocated work groups created by OpenCL framework

is then divided into warps (hardware blocks with 32 threads). Execution instructions of these warps on blocks of processors are provided using warp schedulers dynamically. Global memory accesses in warp should be coalesced. Otherwise, additional memory operations are executed. The local memory is organized into banks. Access to the same banks from warp causes serialization. This issue is referred to as a bank conflict. The serialization of local memory operations and uncoalesced global memory access can cause a performance degradation.

This paper is further focused on the OpenCL framework.[1] OpenCL is a framework for general-purpose parallel programming across multiple device types. In this framework, a platform independent executable program is called the kernel. The kernel is executed on required number of threads that identify their data and control flow by their N-dimensional indices. These threads are organized into work groups with identical user-defined number of threads. The threads in such a group can cooperate with each other through local memory and barriers.

Several methods for the 2-D DWT computation using GPU have been published in the last decade. For example, the simplest row–column methods transform the whole image at once. Usually, the transposition is needed between the horizontal and vertical part. Furthermore, the block-based methods transform the image using smaller blocks utilizing the row-column method inside. Unfortunately, such a method results in several independent transforms instead a single seamless one. Finally, the pipelined methods transform the image using column strips while employing the sliding window on them.

In this paper, we propose two novel block-based methods computing the seamless 2-D transform. The first of them employs the separable (row–column) lifting scheme inside the overlapping blocks. The second uses a non-separable lifting scheme recently proposed. Both of the proposed methods consistently outperform the existing methods.

The rest of the paper is organized as follows. The Related Work section summarizes the state of the art, especially existing GPU implementations. The heart of our work is presented in Block-Based Approach section. First, we propose the separable transform. Further in the text, the non-separable method is discussed. Finally, Conclusions section summarizes the paper and outlines the future work.

2 Related Work

This section takes a closer look at the discrete wavelet transform and revises the state of the art of its implementation on contemporary graphics cards.

The DWT can be understood as a transform suitable for decomposition of a signal into low-pass and high-pass frequency components. Usually, such a decomposition is performed at several scales resulting in a multi-scale signal representation. At this point, we are considering one-dimensional signals. The transform of 2-D signals is computed through the tensor product of these 1-D transforms. For various requirements, different strategies of 2-D transform computation emerged. Going back to 1-D transform, as the discrete wavelet transform is

[1] `http://www.fit.vutbr.cz/research/prod/index.php?id=434`.

a linear one, the decomposition into the low-pass and high-pass components can be performed through a convolution scheme with two filters. However, the more efficient computational scheme according to the number of arithmetic operations exists. This scheme is referred to as the lifting scheme. Additionally, using this scheme, the whole signal can be transformed in-place. Specifically, any discrete wavelet transform can be factored into a finite sequence of lifting steps. These steps alternately update odd and even intermediate results using short FIR (finite impulse response) filters. When evaluating this scheme, intermediate results can be appropriately shared between neighbouring coefficients.

The discrete wavelet transform is often used as a basis for sophisticated compression algorithms. This paper focuses on a popular CDF (Cohen-Daubechies-Feauveau) 9/7 wavelet. This wavelet is used, e.g., in JPEG 2000 image compression standard. In [2], Daubechies and Sweldens factored CDF 9/7 wavelet into four successive lifting steps, employing short symmetric two-taps FIR filters. A data-flow diagram of the factorization (without scaling) is depicted in Fig. 1, where, the $\alpha, \beta, \gamma, \delta$ are real constants specific to CDF 9/7 transform. Formally, the forward transform in Fig. 1 can be expressed by the dual polyphase matrix

$$\tilde{P}(z) = \begin{bmatrix} 1 & \alpha\left(1+z^{-1}\right) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \beta\left(1+z\right) & 1 \end{bmatrix} \begin{bmatrix} 1 & \gamma\left(1+z^{-1}\right) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \delta\left(1+z\right) & 1 \end{bmatrix}. \quad (1)$$

In this paper, we consider the lifting scheme only, as it is usually a better alternative. A detailed comparison of the convolution and lifting schemes on GPUs was addressed, e.g., in [11] and [12].

The implementation of this transform was comprehensively studied on various platforms including the modern GPUs. Considering this scenario, the input image has to be initially transferred from main memory into memory on the graphics card. Similarly, the resulting coefficients could be transferred back. Having the input 2-D image in the GPU global memory, different strategies of 2-D DWT implementation can be used. These strategies can be divided into three groups – row–column, block-based, and pipelined methods.

The row–column method applied on the entire 2-D image was used for instance in [11], [12], [3], [1], [4], [5]. In [3] and [1], data transposition was performed in between the horizontal and vertical series of 1-D transforms. In [11] and [12], Tenllado *et al.* adapted the discrete wavelet transform on GPU fragment shaders. As this paper is focused on the OpenCL framework, we will not discuss their paper in more details. The other cited papers are focused on the CUDA architecture. In [3], the convolution scheme is applied on each row. Then, the image matrix is transposed and the convolutions are applied on each column. Finally, the image is transposed back. In [4] and [5], V. Galiano *et al.* compared several CUDA implementations of DWT. They used the CDF 9/7 wavelet and convolution-based algorithm on entire rows/columns. Their fastest implementation uses the coalesced memory access.

In [1], the authors calculate the wavelet transform through 4 kernels. The first kernel performs an image transposition using work groups of size 16×16 threads, where the thread processes one image element. To ensure coalesced

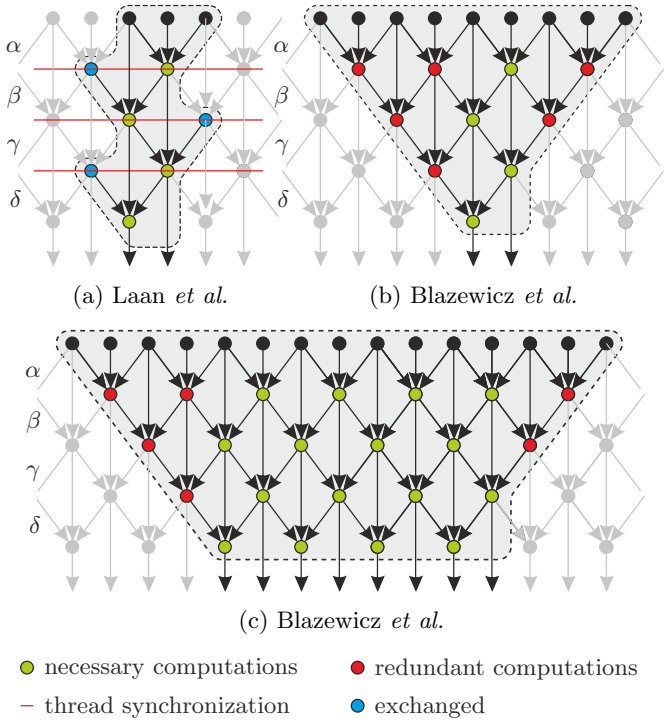


Fig. 1: A portion of the data-flow graph attributable to individual threads. The method of (a) Laan and two methods used by Blazewicz – with (b) one, and (c) four pairs.

global memory access, the transposition in the shared memory is used rather than directly in the global memory. In the second kernel, the vertical wavelet transform is performed as follows. Each thread loads its elements from the global memory and stores them into the shared memory. Then, the adjacent elements, that are required for the computation of the output coefficient, are loaded from the shared memory into registers. The threads compute their output coefficients using 4 steps of the wavelet scheme independently to each other (with no synchronization). When the computation is finished, the output coefficients are written back to the global memory. The third and the fourth kernels calculate the image transposition and the horizontal wavelet transform in the same way as the first two kernels. The calculations that are performed by a single thread using the approach described can be seen in Fig. 1b and Fig. 1c.

The pipelined approach was used in [8] and [9]. In [8], Laan *et al.* accelerated the Dirac video codec using the CUDA platform. In [9], the authors provided a detailed analysis of the DWT implementation using the lifting scheme on the CUDA platform. They focused on 2-D and 3-D methods of DWT implementation using several wavelets including CDF 9/7. In the horizontal part of their transform, each work group is mapped to a single image row. Each thread com-

putes one coefficient per a single step and shares it with other threads. Because of non-atomic instructions issued in whole group, memory barrier is needed in between each two steps. See Fig. 1a. The vertical part of their transform maps each work group to multiple vertical strips with a width that ensures coalesced global memory accesses and bank-conflict-free shared memory transfers.

Another row-column approach was used in [7]. The horizontal transform is computed in the same way as Blazewicz *et al.* did. The vertical transform is computed using 32 coefficients wide strips per work group like in Laan's implementation. The difference between the Laan's and Kucis's vertical methods lies in processing assigned to a single thread. Kucis *et al.* used rotated Blazewicz's approach with 2 pairs per thread mapping. Moreover, Kucis *et al.* also demonstrated that their approach outperforms Laan's and Blazewicz's ones. The approach of Kucis is used as a reference approach and labeled as *Kucis2014*.

The approaches in [10] as well as [1] are focused on the lifting scheme. Their implementations split the image into small tiles and perform several independent transforms on each of them. Thus, they performed several independent transforms (introducing a block effect) which is different and much easier task comparing to what we are dealing with in this paper.

As it can be seen, the problem of the efficient 2-D discrete wavelet transform implementation on conventional GPUs was fairly well studied. However, we see several gaps which can allow for additional speedups. Specifically, only the separable 2-D schemes were examined so far. These schemes require to pass the results through the global memory, while causing unnecessary memory traffic.

3 Block-Based Approach

The heart of our work is presented in this section. At the beginning, we propose the separable block-based method. Afterwards, we discuss the block-based method utilizing non-separable lifting scheme recently proposed. The performance comparison of the proposed methods is shown in Fig. 4. As it can be seen, the block-based methods perform consistently faster compared to the best of the existing methods. Our implementation is based on the OpenCL framework. All of the algorithms are evaluated using AMD R9 290X and NVIDIA TitanX graphics cards. The main benefit of the block-based methods is the reduction of memory access count as the data is read as well as written only once.

3.1 Separable Method

Except for the sliding window, our separable block-based approach uses the same scheme as the Laan's method. The threads in each work group are responsible for processing of 2×2 input coefficients. At the beginning, the thread loads their coefficients from the global memory and stores them into separate shared memory locations. The computation is briefly illustrated in Fig. 2. In the first step of the horizontal pass, each of the threads computes the LH coefficient using two LL coefficients of the thread itself and the thread on the right. Additionally,

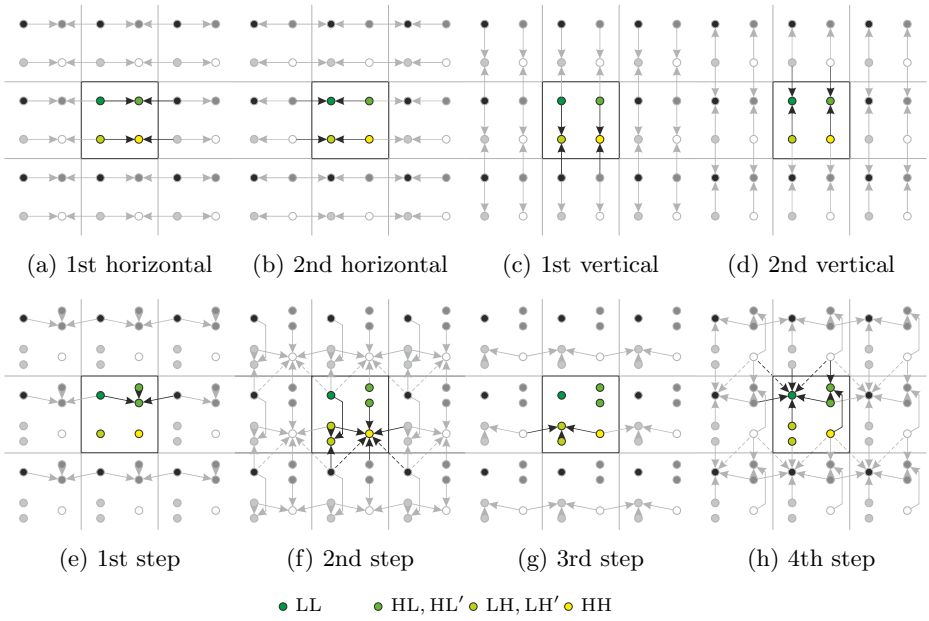


Fig. 2: The separable (top) and non-separable block-based approach (bottom).

the HH coefficient is computed using HL coefficients of the current thread and the thread on the right. In the second step, the computation of LL and HL coefficients is performed in the same way as the computation of the LH and HH coefficients. After that, these two steps are repeated with a substitution of α and β with γ and δ coefficients.

The vertical steps are performed in the same way as the horizontal steps except for a rotation of the scheme by 90 degrees. Unlike the horizontal pass, synchronization using the memory barrier is required between the steps. Horizontal steps are synchronization-free thanks to the atomicity of hardware instructions. Fig. 2 shows individual steps of the underlying data-flow graph.

3.2 Non-Separable Method

In [6], the authors derived a non-separable 2-D lifting scheme for CDF 5/3 and subsequently CDF 9/7 transforms. As initial step of CDF 5/3 transform, the input signal is split into quadruples (LL, HL, LH, HH). Then, lifting steps leading to the calculation HH coefficients are performed. This is followed by parallel computation of the HL and LH coefficients. In the third step, the LL coefficient is updated. Finally, the coefficients can be scaled. The scheme for CDF 9/7 comprises two these connected transforms.

Motivated by the work of Iwahashi *et al.* [6], we have reorganized the elementary lifting FIR filters in order to obtain a highly parallelizable scheme suitable for the modern GPUs. The main purpose of this modification is to minimize

the number of memory barriers that slow down the calculation. As a result, we get several non-separable two-dimensional FIR filters. For their description, we employ the well known z-transform notation. The transfer function of the two-dimensional FIR filter $x(k_m, k_n)$ is defined as

$$X(z_m, z_n) = \sum_{k_m=-\infty}^{\infty} \sum_{k_n=-\infty}^{\infty} x(k_m, k_n)\, z_m^{-k_m} z_n^{-k_n}, \tag{2}$$

where m refers to the horizontal axis and n to the vertical one. Moreover, to keep consistency with [6], the $H^*(z_m, z_n) = H(z_n, z_m)$ denotes a filter transposed to the $H(z_m, z_n)$. Furthermore, the $\overline{H}(z_m, z_n) = H(z_n^{-1}, z_m^{-1})$ denotes a filter reversed along the m- as well as n-axis. Coupled together, the $\overline{H}^*(z_m, z_n)$ denotes a transposed and reversed filter to the original $H(z_m, z_n)$. The scheme we formed is composed of three elementary filters F, G, H given by

$$\begin{bmatrix} \mathrm{F_a} \\ \mathrm{G_a} \\ \mathrm{H_a} \end{bmatrix} = \begin{bmatrix} F_a(z_m, z_n) \\ G_a(z_m, z_n) \\ H_a(z_m, z_n) \end{bmatrix} = a \begin{bmatrix} 1 \\ z_n \\ 1 + z_m \end{bmatrix}, \tag{3}$$

where a denotes a filter parameter. The filters above are assembled into more complex operations. Our scheme consists of two halves between which a memory barrier is placed. The first half of the scheme uses the following filters. Similarly, the second half uses these filters in the reverse orientation. Due to the limited place, we have made a small abuse of notation. Instead of the full notation $H(z_m, z_n)$, we only use a shortened labeling, such as H.

$$\begin{bmatrix} \mathrm{F_a} \\ \mathrm{G_a} \\ \mathrm{H_a} \\ \mathrm{H_a^*} \\ \mathrm{G_a H_a} \end{bmatrix} = \begin{bmatrix} a \\ a\, z_n \\ a\,(1 + z_m) \\ a\,(1 + z_n) \\ a^2\,(z_n + z_m z_n) \end{bmatrix}, \qquad \begin{bmatrix} \overline{\mathrm{F}}_\mathrm{a} \\ \overline{\mathrm{G}}_\mathrm{a} \\ \overline{\mathrm{H}}_\mathrm{a} \\ \overline{\mathrm{H}}_\mathrm{a}^* \\ \overline{\mathrm{G}}_\mathrm{a} \overline{\mathrm{H}}_\mathrm{a} \end{bmatrix} = \begin{bmatrix} a \\ a\, z_n^{-1} \\ a\,(1 + z_m^{-1}) \\ a\,(1 + z_n^{-1}) \\ a^2\,(z_n^{-1} + z_m^{-1} z_n^{-1}) \end{bmatrix} \tag{4}$$

Finally, our scheme is composed of four steps referred to as S^1 to S^4. Between the second S^2 and the third S^3 step, the memory barrier must be inserted in order to properly exchange intermediate results. Additionally, our scheme requires the induction of two auxiliary variables per each quadruple of coefficients LL, HL, LH, and HH. These are denoted as $\mathrm{HL}', \mathrm{LH}'$. This is valid regardless of their initial as well as final values. The scheme

$$\mathbf{y} = S_\beta^4 S_\beta^3 S_\alpha^2 S_\alpha^1 \mathbf{x} \tag{5}$$

describes the relation between input $\mathbf{x}$ and output $\mathbf{y}$ vectors

$$\left[\ \mathrm{LL\ HL\ LH\ HH\ HL'\ LH'}\ \right]^T. \tag{6}$$

Each single thread of the work group is responsible of one such a vector.

Regarding this notation, the individual steps are defined as follows. For better understanding, the signal-processing block diagram of this scheme is shown in Fig. 3. In addition, the operations are graphically illustrated in Fig. 2.

$$S_{\alpha}^{1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ \mathrm{H}_{\alpha} & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

$$S_{\alpha}^{2} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \mathrm{G}_{\alpha}\mathrm{H}_{\alpha} & \mathrm{G}_{\alpha} & \mathrm{H}_{\alpha} & 1 & \mathrm{F}_{\alpha} & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ \mathrm{H}_{\alpha}^{*} & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{8}$$

$$S_{\beta}^{3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \overline{\mathrm{H}}_{\beta} & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

$$S_{\beta}^{4} = \begin{bmatrix} 1 & 0 & \overline{\mathrm{F}}_{\beta} & \overline{\mathrm{G}}_{\beta}\overline{\mathrm{H}}_{\beta} & \overline{\mathrm{H}}_{\beta} & \overline{\mathrm{G}}_{\beta} \\ 0 & 0 & 0 & \overline{\mathrm{H}}_{\beta}^{*} & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

Compared to [6], the total number of arithmetic operations has been reduced form 24 to 20. The calculation of CDF 9/7 transform comprises two of these connected transforms (the first one with α, β, the second with γ, δ) between them another barrier is placed. In total, the calculation contains 3 memory barriers.

4 Conclusions

We have presented two novel block-based approaches to 2-D wavelet transform using modern GPUs. These approaches can handle high resolution images while producing the seamless transform. Both of the proposed methods consistently outperform the existing methods with all tested GPUs.

The first presented approach utilizes classical separable 2-D lifting scheme. Whereas the second approach employs a novel 2-D non-separable scheme. Considering the second one, we have minimized the number of memory barriers. Moreover, as compared to the existing non-separable scheme, the total number of arithmetic operations has been reduced form 24 to 20.

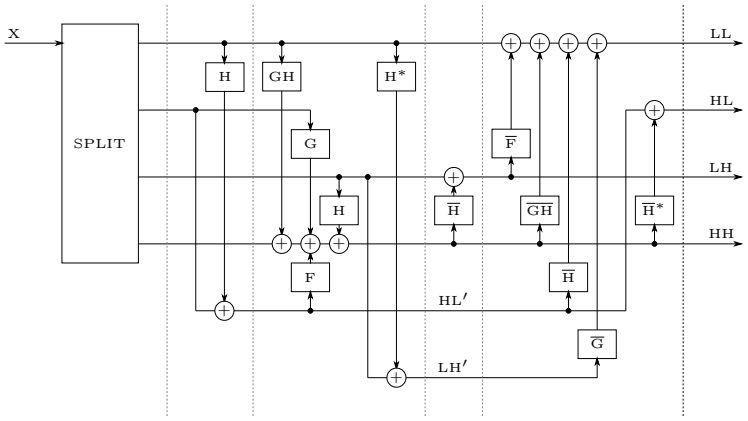
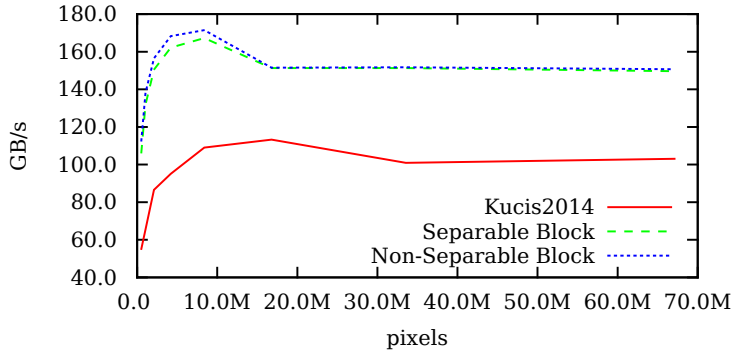
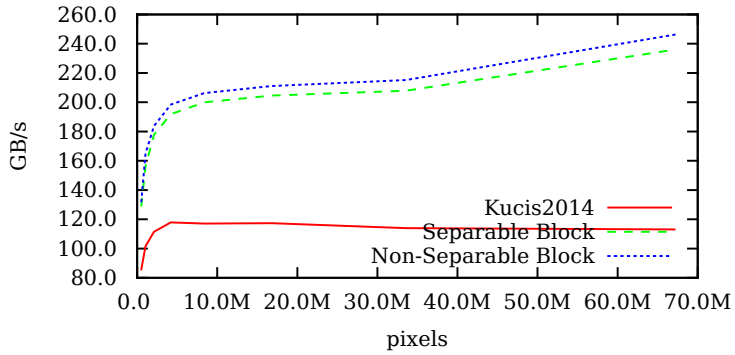


Fig. 3: Block diagram of the proposed non-separable scheme.



(a) AMD R9 290X



(b) NVIDIA TitanX

Fig. 4: Throughput performance. *Kucis2014* is the reference state-of-the-art method.

The future work includes behavior of the proposed methods under a multi-scale decomposition. Another direction of our research may include a connection with some practical application (e.g., JPEG 2000 scheme).

Acknowledgement

This work has been supported by the TACR Competence Centres project V3C – Visual Computing Competence Center (no. TE01020415).

References

1. Błażewicz, M., Ciżnicki, M., Kopta, P., Kurowski, K., Lichocki, P.: Two-dimensional discrete wavelet transform on large images for hybrid computing architectures: GPU and CELL. In: Euro-Par 2011: Parallel Processing Workshops, LNCS, vol. 7155, pp. 481–490. Springer (2012)
2. Daubechies, I., Sweldens, W.: Factoring wavelet transforms into lifting steps. Journal of Fourier Analysis and Applications 4(3), 247–269 (1998)
3. Franco, J., Bernabe, G., Fernandez, J., Acacio, M.: A parallel implementation of the 2D wavelet transform using CUDA. In: 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing. pp. 111–118 (2 2009)
4. Galiano, V., López, O., Malumbres, M., Migallón, H.: Improving the discrete wavelet transform computation from multicore to GPU-based algorithms. In: Proceedings of the 11th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE). pp. 544–555 (2011)
5. Galiano, V., López, O., Malumbres, M., Migallón, H.: Parallel strategies for 2D discrete wavelet transform in shared memory systems and GPUs. The Journal of Supercomputing 64(1), 4–16 (2013)
6. Iwahashi, M., Kiya, H.: Non separable two dimensional discrete wavelet transform for image signals. In: Discrete Wavelet Transforms – A Compendium of New Approaches and Recent Applications. InTech (2013)
7. Kucis, M., Barina, D., Kula, M., Zemcik, P.: 2-D discrete wavelet transform using GPU. In: 5th Workshop on Application for Multi-Core Architectures. pp. 1–6. IEEE Computer Society (2014)
8. van der Laan, W., Roerdink, J.B.T.M., Jalba, A.: Accelerating wavelet-based video coding on graphics hardware using CUDA. In: Proceedings of 6th International Symposium on Image and Signal Processing and Analysis. pp. 608–613 (Sep 2009)
9. van der Laan, W.J., Jalba, A.C., Roerdink, J.B.T.M.: Accelerating wavelet lifting on graphics hardware using CUDA. IEEE Transactions on Parallel and Distributed Systems 22(1), 132–146 (2011)
10. Matela, J.: GPU-based DWT acceleration for JPEG2000. In: Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science. pp. 136–143 (2009)
11. Tenllado, C., Lario, R., Prieto, M., Tirado, F.: The 2D discrete wavelet transform on programmable graphics hardware. In: Visualization, Imaging and Image Processing Conference 2004. pp. 808–813 (9 2004)
12. Tenllado, C., Setoain, J., Prieto, M., Pinuel, L., Tirado, F.: Parallel implementation of the 2D discrete wavelet transform on graphics processing units: Filter bank versus lifting. IEEE Transactions on Parallel and Distributed Systems 19(3), 299–310 (2008)