Accelerating Discrete Wavelet Transforms on Parallel Architectures

David Barina Michal Kula Michal Matysek Pavel Zemcik

Centre of Excellence IT4Innovations
Faculty of Information Technology
Brno University of Technology
Bozetechova 1/2, Brno
Czech Republic
{ibarina,ikula,imatysek,zemcik}@fit.vutbr.cz

ABSTRACT

The 2-D discrete wavelet transform (DWT) can be found in the heart of many image-processing algorithms. Until recently, several studies have compared the performance of such transform on various shared-memory parallel architectures, especially on graphics processing units (GPUs). All these studies, however, considered only separable calculation schemes. We show that corresponding separable parts can be merged into non-separable units, which halves the number of steps. In addition, we introduce an optional optimization approach leading to a reduction in the number of arithmetic operations. The discussed schemes were adapted on the OpenCL framework and pixel shaders, and then evaluated using GPUs of two biggest vendors. We demonstrate the performance of the proposed non-separable methods by comparison with existing separable schemes. The non-separable schemes outperform their separable counterparts on numerous setups, especially considering the pixel shaders.

Keywords
Discrete wavelet transform, Image processing, Synchronization, Graphics processors

1 INTRODUCTION

The discrete wavelet transform became a very popular image processing tool in last decades. A widespread use of this transform has resulted in a development of fast algorithms on all sorts of computer systems, including shared-memory parallel architectures. At present, the GPU is considered as a typical representative of such parallel architectures. In this regard, several studies have compared the performance of various 2-D DWT computational approaches on GPUs. All of these studies are based on separable schemes, whose operations are oriented either horizontally or vertically. These schemes comprise the convolution and lifting. The lifting requires fewer arithmetic operations as compared with the convolution, at the cost of introducing some data dependencies. The number of operations should be proportional to a transform performance. However, also the data dependencies may form a bottleneck, especially on shared-memory parallel architectures.

In this paper, we show that the fastest scheme for a given architecture can be obtained by fusing the corresponding parts of the separable schemes into new structures. Several new non-separable schemes are obtained in this way. More precisely, the underlying operations of these schemes can be associated with neither horizontal nor vertical axes. In addition, we present an approach where each scheme can be adapted to a particular platform in order to reduce the number of operations. This possibility was completely omitted in existing studies. Our reasoning is supported by extensive experiments on GPUs using OpenCL and pixel shaders (fragment shaders in OpenGL terminology). The presented schemes are general, and they are not limited to any specific type of DWT. To clarify the situation, they all compute the same values.

The rest of this paper is organized as follows. Section Background formally introduces the problem definition. Section Related Work briefly presents the existing separable approaches. Section Proposed Schemes presents the proposed non-separable schemes. Section Optimization Approach discusses the optimization approach that reduces the number of operations. Section Evaluation evaluates the performance on GPUs in the pixel shaders and OpenCL framework. Eventually, Section Conclusions closes the paper. This section is followed by Section Appendix for readers not familiar with signal-processing notations.

2 BACKGROUND

Since the separable schemes are built on the one-dimensional transform, a widely-used z-transform is used for the description of underlying FIR filters. The transfer function of the filter (g_k) is the polynomial

$$G(z) = \sum_k g_k z^{-k},$$

where the k refers to the time axis. Below in the text, the one-dimensional transforms are used in conjunction with two-dimensional signals. For this case, the transfer function of the filter $\left(g_{k_m,k_n}\right)$ is defined as the bivariate polynomial

$$G(z_m, z_n) = \sum_{k_m} \sum_{k_n} g_{k_m,k_n} z_m^{-k_m} z_n^{-k_n},$$

where the subscript m refers to the horizontal axis and n to the vertical one. The $G^*(z_m, z_n) = G(z_n, z_m)$ is a polynomial transposed to a polynomial $G(z_m, z_n)$. A shortened notation G is only written in order to keep the notation readable.

A discrete wavelet transform is a signal-processing tool which is suitable for the decomposition of a signal into low-pass and high-pass components. In detail, the single-scale transform splits the input signal into two components, according to a parity of its samples. Therefore, the DWT is described by 2×2 matrices. As shown by Mallat [10], the transform can be computed by a pair of filters followed by subsampling by a factor of 2. The filters are referred to as $\mathrm{G}_0, \mathrm{G}_1$. The transform can also be represented by the polyphase matrix

$$\begin{bmatrix} \mathrm{G}_1{}^{(o)} & \mathrm{G}_1{}^{(e)} \\ \mathrm{G}_0{}^{(o)} & \mathrm{G}_0{}^{(e)} \end{bmatrix}, \qquad (1)$$

where the polynomials $\mathrm{G}^{(e)}$ and $\mathrm{G}^{(o)}$ refer to the even and odd terms of G. This polyphase matrix defines the convolution scheme. To avoid misunderstandings, it is necessary to say that, in this paper, column vectors are transformed to become another columns. For example, $\mathrm{y} = \mathrm{Mx}$ and $\mathrm{y} = \mathrm{M}_2\mathrm{M}_1\mathrm{x}$ are transforms represented by one and two matrices, respectively. Following the algorithm by Sweldens [14, 4], the convolution scheme in (1) can be factored into a sequence

$$\prod_k \begin{bmatrix} 1 & \mathrm{U}^{(k)} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \mathrm{P}^{(k)} & 1 \end{bmatrix} \qquad (2)$$

of K pairs of short filterings, known as the lifting scheme. The filters employed in (2) are referred to as the lifting steps. Usually, the first step $\mathrm{P}^{(k)}$ in the kth pair is referred to as the predict and the second one $\mathrm{U}^{(k)}$ as the update. The lifting scheme reduces the number of operations by up to half. Since this paper is mostly focused on a single pair of steps, the superscript (k) is omitted in the text below. Note that the number of operations is calculated as the number of distinct (in a column) terms of all polynomials in all matrices, excluding units on diagonals.

Considering the shared-memory parallel architectures, the processing of single or several samples is mapped to independent processing units. In order to avoid race conditions during data exchange, the units must use some synchronization method (barrier). In the lifting scheme, the barriers are required before the lifting steps. In the convolution scheme, the barrier is only required before starting the calculation. In this paper, the barriers are indicated by the | symbol. For example, $\mathrm{M}_2|\mathrm{M}_1$ are two adjacent lifting steps separated by the barrier. For simplicity, the number of barriers is also called the number of steps in the text below.

The 2-D transform is defined as a tensor product of 1-D transforms. Consequently, the transform splits the signal into a quadruple of wavelet coefficients. Therefore, the 2-D DWT is described by 4×4 matrices. See Section Appendix for details. Following the pioneering paper of Mallat [10], the 1-D transforms are applied in both directions sequentially. By its nature, this scheme can be referred to as the separable convolution. The calculations in a single direction are performed in a single step. This means two steps for the two dimensions. The scheme can formally be described as

$$\mathbf{N}^V \,|\, \mathbf{N}^H \,|,$$

where $\mathbf{N}^H$ and $\mathbf{N}^V$ are 1-D transforms in horizontal and in vertical direction. For the well-known Cohen-Daubechies-Feauveau (CDF) wavelet with 9/7 samples, such as used in the JPEG 2000 standard, these matrices are graphically illustrated in Figure 1. Here, only the horizontal part is shown. Particularly, the filters in the figure are of sizes 9 and 7 taps. The 🔴, 🟢, 🔵, and ⚫ circles represent the quadruple of wavelet coefficients. Figures shown are for illustration purpose only.

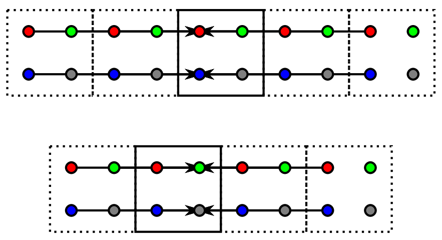


Figure 1: Horizontal part of the separable convolution scheme for the CDF 9/7 wavelet. Two appropriately chosen pairs of matrix rows are depicted in separate subfigures. The arrows are pointing to the destination operand and denote a multiply–accumulate operation, with multiplication by a real constant. The arrows in the same row overlap.

Another scheme used for 2-D transform is the separable lifting. Similarly to the previous case, the predict and update lifting steps can be applied in both directions sequentially. Moreover, horizontal and vertical steps can be arbitrarily interleaved thanks to the linear nature of the filters. Therefore, the scheme is defined as

$$\mathrm{S}_\mathrm{U}^V \,|\, \mathrm{S}_\mathrm{U}^H \,|\, \mathrm{T}_\mathrm{P}^V \,|\, \mathrm{T}_\mathrm{P}^H \,|,$$

wherein the predict steps T always precede the update steps S. The above mapping corresponds to a single P and U pair of lifting steps. For multiple pairs, the scheme is separately applied to each such pair. In order to describe 2-D matrices, the lifting steps must be extended into two dimensions as

$$\begin{bmatrix} \mathrm{G} \\ \mathrm{G}^* \end{bmatrix} = \begin{bmatrix} G\ (z_m, z_n) \\ G^*(z_m, z_n) \end{bmatrix} = \begin{bmatrix} G(z_m) \\ G(z_n) \end{bmatrix}.$$

Then, the individual steps are defined as

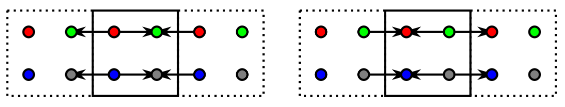
$$\mathrm{T}_\mathrm{P}^H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \mathrm{P} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \mathrm{P} & 1 \end{bmatrix},$$

$$\mathrm{T}_\mathrm{P}^V = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \mathrm{P}^* & 0 & 1 & 0 \\ 0 & \mathrm{P}^* & 0 & 1 \end{bmatrix},$$

$$\mathrm{S}_\mathrm{U}^H = \begin{bmatrix} 1 & \mathrm{U} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \mathrm{U} \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\mathrm{S}_\mathrm{U}^V = \begin{bmatrix} 1 & 0 & \mathrm{U}^* & 0 \\ 0 & 1 & 0 & \mathrm{U}^* \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

For the CDF wavelets, the matrices are also illustrated in Figure 2, again showing the horizontal part only.



(a) T_P^H (b) S_U^H

Figure 2: The horizontal part of the separable lifting scheme for the CDF wavelets.

3 RELATED WORK

This section briefly reviews papers that motivated our research. So far, several papers have compared the performance of the separable lifting and separable convolution schemes on GPUs. Especially, Tenllado *et al.* [15] compared these schemes on GPUs using pixel shaders. The authors mapped data to 2-D textures, constituted by four floating-point elements. They concluded that the separable convolution is more efficient than the separable lifting scheme in most cases. They further noted that fusing several consecutive kernels might significantly speed up the execution, even if the complexity of the resulting fused pixel program is higher.

Kucis *et al.* [8] compared the performance of several recently published schedules for computing the 2-D DWT using the OpenCL framework. All of these schedules use separable schemes, either the convolution or lifting. In more detail, the work compares a convolution-based algorithm proposed in [5] against several lifting-based methods [2, 16] in the horizontal part of the transform. The authors concluded that the lifting-based algorithm of Blazewicz *et al.* [2] is the fastest method. Furthermore, Laan *et al.* [16] compared the performance of their separable lifting-based method against a convolution-based method. They concluded that the lifting is the fastest method. The authors also compared the performance of implementations in CUDA and pixel shaders, based on the work of Tenllado [15]. The CUDA implementation proved to be the faster choice. In this regard, the authors noted that a speedup in CUDA occurs because the CUDA effectively makes use of on-chip memory. This use is not possible in pixel shaders, which exchange the data using off-chip memory. Other important separable approaches can be found in [11, 6, 13, 12].

This paper is based on the previous works in [1, 9]. In those works, we introduced several non-separable schemes for calculation of 2-D DWT. However, we have not considered important structures, such as polyconvolutions. We contribute this consideration with this paper. Moreover, differences and similarities between the separable schemes and their non-separable counterparts are homogeneously discussed here. All these schemes are also thoroughly analyzed and evaluated.

Considering the present papers, we see that a possible fusion of separable parts into new non-separable structures is not considered. Therefore, we investigate on this promising technique in the following sections.

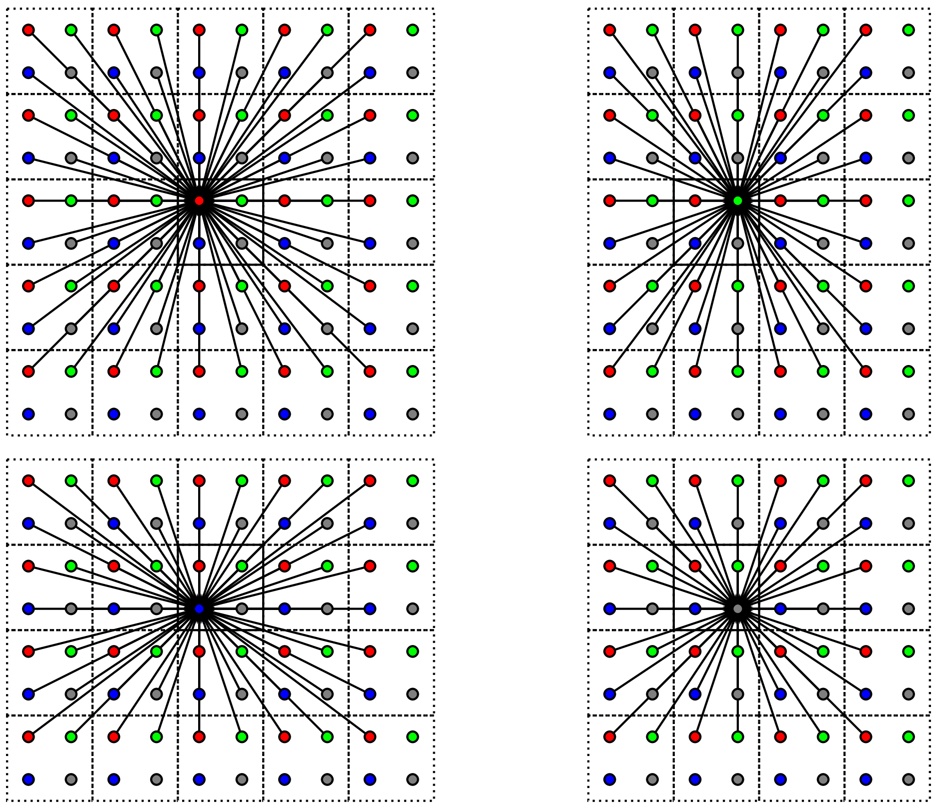


Figure 3: Non-separable convolution scheme for the CDF 9/7 wavelet. The individual rows of N are depicted in separate subfigures. The sizes are from top to bottom and left to right: 9×9, 7×9, 9×7, 7×7.

4 PROPOSED SCHEMES

As stated above, the existing approaches did not study the possibility of a partial fusion of lifting polyphase matrices. This section presents three alternative non-separable schemes for the calculation of the 2-D transform. The contribution of this paper starts with this section. To avoid confusion, please note that the proposed schemes compute the same values as the original ones.

The non-separable convolution scheme is a counterpart to the separable convolution. Unlike the separable scheme, all horizontal and vertical calculations are performed in a single step

$$\mathbf{N}\,\Big|,$$

where $\mathbf{N} = \mathbf{N}^V\mathbf{N}^H$ is a product of 1-D transforms in horizonal and vertical directions. The drawback of this scheme is that it requires the highest number of arithmetic operations. For the CDF 9/7 wavelet, the matrix is graphically illustrated in Figure 3. Here, the 2-D filters are of sizes 9×9, 7×9, 9×7, and 7×7. These sizes make the calculation computationally demanding. Aside from the GPUs, this approach was earlier discussed in Hsia *et al.* [7].

In order to reduce computational complexity, it would be a good idea to construct some smaller non-separable steps. Indeed, the non-separable convolution can be broken into smaller units, referred here to as the non-separable polyconvolutions. For a single pair of lifting steps, the scheme follows from the mapping

$$\mathrm{N}_{\mathrm{P,U}}\,\Big|,$$

where

$$\mathrm{N}_{\mathrm{P,U}} = \begin{bmatrix} \mathrm{V^*V} & \mathrm{V^*U} & \mathrm{U^*V} & \mathrm{U^*U} \\ \mathrm{V^*P} & \mathrm{V^*} & \mathrm{U^*P} & \mathrm{U^*} \\ \mathrm{P^*V} & \mathrm{P^*U} & \mathrm{V} & \mathrm{U} \\ \mathrm{P^*P} & \mathrm{P^*} & \mathrm{P} & 1 \end{bmatrix}$$

and $\mathrm{V} = \mathrm{PU}+1$. For the CDF wavelets, the scheme is graphically illustrated in Figure 4. In this case, the employed filters are of sizes 5×5, 3×5, 5×3, and 3×3. Note that only half of the operations are required specifically for the CDF 9/7 wavelet, compared to the non-separable convolution. For the sake of completeness, it should be pointed out that it is also possible to formulate the separable polyconvolution scheme. In our experiments, this one was however not proven to be useful concerning the performance.

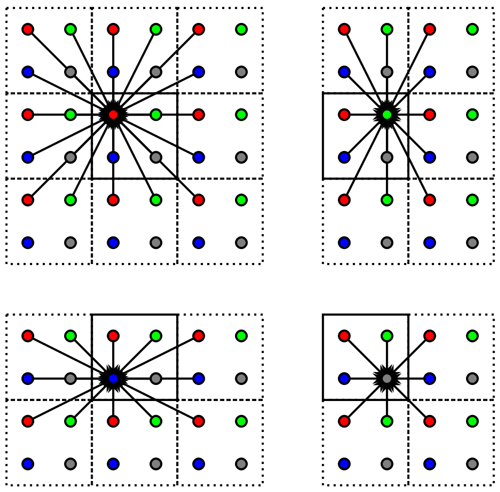


Figure 4: Non-separable polyconvolution scheme for the CDF wavelets. The individual rows of N are depicted in separate subfigures.

By combining the corresponding horizontal and vertical steps of the separable lifting scheme, the non-separable lifting scheme is formed. The number of operations has slightly been increased. The scheme consists of a spatial predict and spatial update step, thus two steps in total for each pair of the original lifting steps. Formally, for each pair of P and U, the scheme follows from

$$\mathrm{S_U} \,|\, \mathrm{T_P} \,|,$$

where

$$\mathrm{T_P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \mathrm{P} & 1 & 0 & 0 \\ \mathrm{P}^* & 0 & 1 & 0 \\ \mathrm{PP}^* & \mathrm{P}^* & \mathrm{P} & 1 \end{bmatrix},$$

$$\mathrm{S_U} = \begin{bmatrix} 1 & \mathrm{U} & \mathrm{U}^* & \mathrm{UU}^* \\ 0 & 1 & 0 & \mathrm{U}^* \\ 0 & 0 & 1 & \mathrm{U} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Note that the spatial filters in PP^* and UU^* may be computationally demanding, depending on their sizes. However, the situation is always better than in the previous two cases. For the CDF wavelets, the scheme is graphically illustrated in Figure 5.

5 OPTIMIZATION APPROACH

This section presents an optimization approach that reduces the number of operations, while the number of steps remains unaffected. Such an approach was not covered in existing studies.

Regardless of the underlying platform, an important observation can be made. A very special form of the operations guarantees that the processing units never access the results belonging to their neighbors. These operations comprise only constants. Since the convolution is a linear operation, the polynomials can be pulled

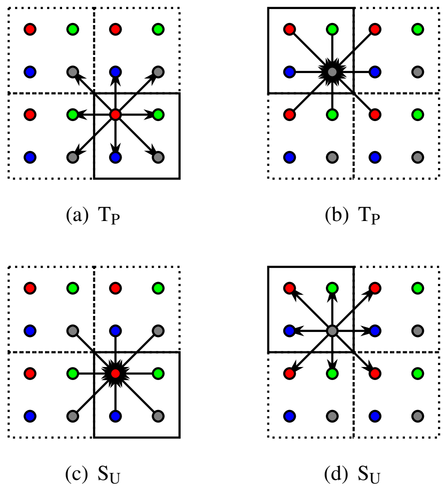


Figure 5: Non-separable lifting scheme for the CDF wavelets.

out of the original matrices, and calculated in a different step. Formally, the original polynomials are split as $\mathrm{P} = \mathrm{P}_0 + \mathrm{P}_1$ and $\mathrm{U} = \mathrm{U}_0 + \mathrm{U}_1$. The P_0 and U_0 are constant. As a next step, the P_0 and U_0 are substituted into the separable lifting scheme. The separable lifting scheme was chosen because it has the lowest number of operations. This part is illustrated in Figure 6. In contrast, the P_1 and U_1 are kept in original schemes. These two steps are then computed without any barrier. The observation is further exploited to adapt schemes for a particular platform.

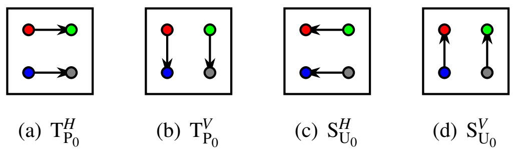


Figure 6: Separable lifting scheme with the polynomials P_0 and U_0.

Now, the improved schemes for the shaders and OpenCL are briefly described. These schemes exploit the above-described observation with the polynomials P_0 and U_0 . On recent GPUs, OpenCL schemes also omit memory barriers due to the SIMD-32 architecture. Note that the non-separable polyconvolution scheme makes sense only when $K > 1$, which is the case of the CDF 9/7 wavelet. Implementations in the pixel shaders map input and output data to 2-D textures. There is no possibility to retain some results in registers, and the results are exchanged through textures in off-chip memory. Considering the OpenCL implementations, a data format is not constrained. The image is divided into overlapping blocks and on-chip memory shared by all threads in a block is utilized to exchange the results. Additionally, some results are passed in registers.

This paper explores the performance for three frequently used wavelets, namely, CDF 5/3, CDF 9/7 [3], and DD 13/7 [14]. Their fundamental properties are listed in Table 1: number of steps and arithmetic operations. Note that the number of operations is commonly proportional to a transform performance. Additionally, the number of steps correspond to the number of synchronizations on parallel architectures, which also form a performance bottleneck.

Table 1: The total number of steps and arithmetic operations for the optimized schemes.

(a) CDF 5/3

scheme		steps	operations	
			OpenCL	shaders
separable	convolution	2	20	22
separable	lifting	4	16	16
non-separable	convolution	1	23	39
non-separable	lifting	2	18	18

(b) CDF 9/7

scheme		steps	operations	
			OpenCL	shaders
separable	convolution	2	56	58
separable	polyconv.	4	40	56
separable	lifting	8	32	32
non-separable	convolution	1	152	200
non-separable	polyconv.	2	46	62
non-separable	lifting	4	36	36

(c) DD 13/7

scheme		steps	operations	
			OpenCL	shaders
separable	convolution	2	60	60
separable	lifting	4	32	32
non-separable	convolution	1	203	228
non-separable	lifting	2	50	50

6 EVALUATION

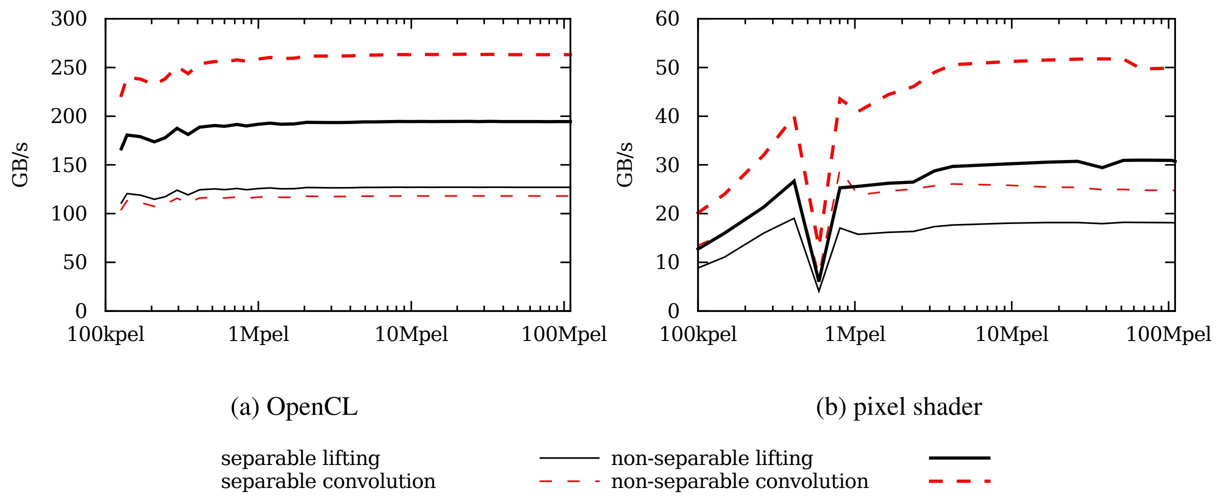
The experiments in this paper were performed on GPUs of the two biggest vendors NVIDIA and AMD using the OpenCL and pixel shaders. In these experiments, only a transform performance was measured, usually in gigabytes per second (GB/s). The host system does not help in the calculation, i.e. with respect to padding or pre/post-processing. Results for only two GPUs are shown for the sake of brevity: AMD Radeon HD 6970 and NVIDIA Titan X. Their technical parameters are summarized in Table 2.

Table 2: Specifications of the evaluated GPUs.

label	AMD 6970	NVIDIA Titan X
model	Radeon HD 6970	Titan X (Pascal)
multiprocessors	24	28
total processors	1 536	3 584
processor clock	880 MHz	1 417 MHz
performance	2 703 GFLOPS	10 157 GFLOPS
memory clock	1 375 MHz	2 500 MHz
bandwidth	176 GB/s	480 GB/s
on-chip memory	32 KiB	96 KiB

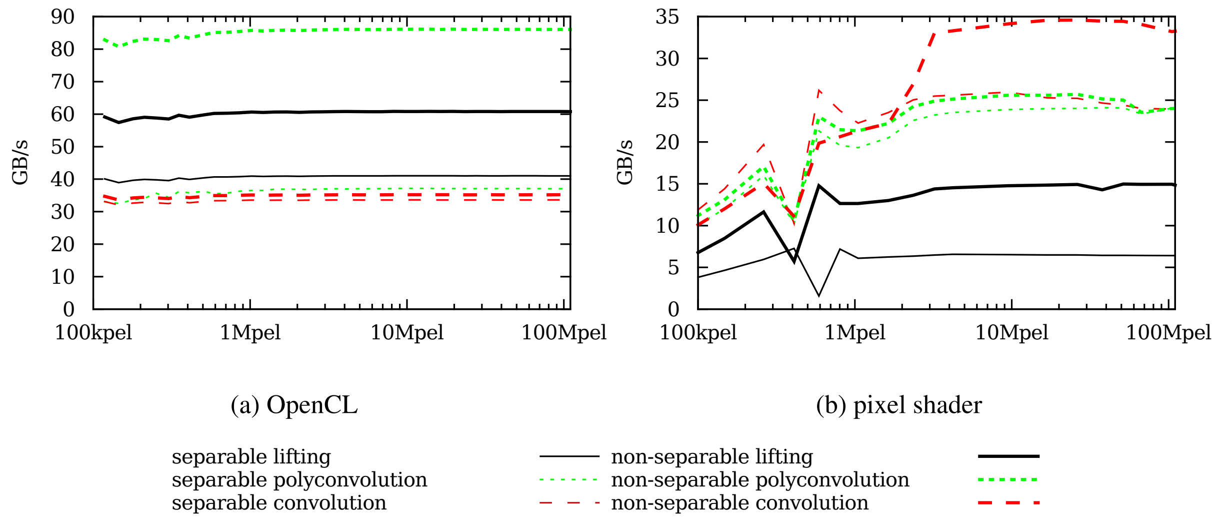
The implementations were created using the DirectX HLSL and OpenCL. The HLSL implementation is used on the NVIDIA Titan X, whereas the OpenCL implementation on the AMD 6970. The results of the performance comparison are shown in Figures 7, 8, and 9. The value on the x-axis is the image resolution in kilo/megapixels (kpel or Mpel). Except for the convolutions for the DD 13/7 wavelet, the non-separable schemes always outperform their separable counterparts. For CDF wavelets, having short lifting filters, the non-separable (poly)convolutions have a better performance than the non-separable lifting scheme. Unfortunately, for the DD 13/7 wavelet, which is characterized by a high number of operations in lifting filters, the results are not conclusive. Considering the implementation in pixel shaders, similar results were also achieved on other GPUs, including NVIDIA unified architectures and AMD GPUs based on Graphics Core Next (GCN) architecture. Whereas for the OpenCL implementation, the non-separable schemes are only proved to be useful for very long instruction word (VLIW) architectures.

Looking at the experiments with the pixel-shader implementations, some transients can be seen at the beginning of the plots (in lower 2 Mpel region). We concluded that these transients are caused by a suboptimal use of cache system, or alternatively by some overhead made by the graphics API. It should be interesting to show some measures provided by an OpenCL profiler. Our profiling revealed that the implementations exhibit only an occupancy 95.24 %. This occupancy is caused by making use of 256 threads in OpenCL work groups and due to maximal number 1344 of threads in multiprocessors (256 times 5 work groups is 1280 out of 1344).



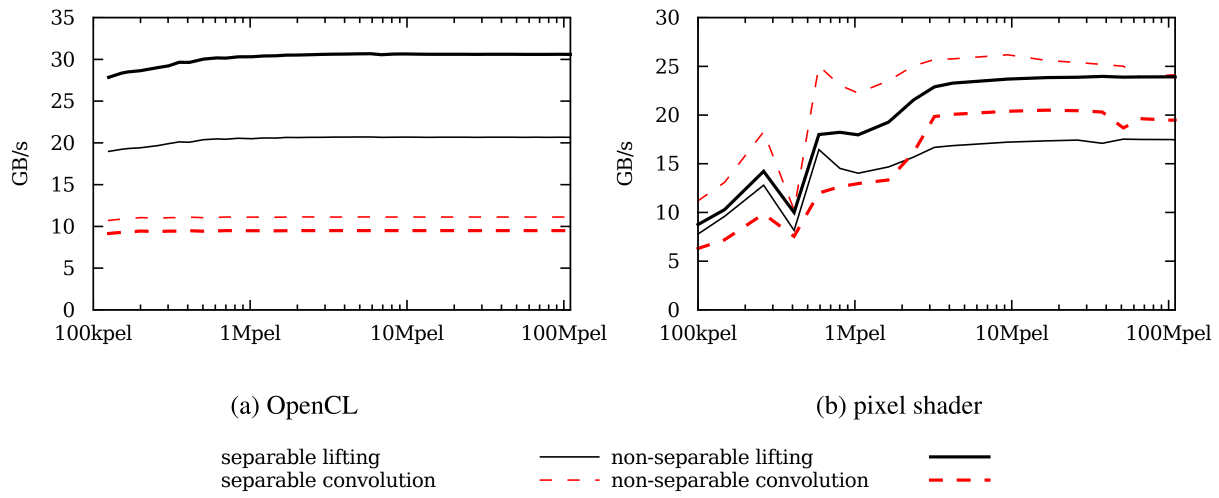
(a) OpenCL (b) pixel shader

Figure 7: Performance for the CDF 5/3 wavelet.



(a) OpenCL (b) pixel shader

Figure 8: Performance for the CDF 9/7 wavelet.



(a) OpenCL (b) pixel shader

Figure 9: Performance for the DD 13/7 wavelet.

7 CONCLUSIONS

This paper presented and discussed several non-separable schemes for the computation of the 2-D discrete wavelet transform on parallel architectures, exemplarily on modern GPUs. As an option, an optimization approach leading to a reduction in the number of operations was presented. Using this approach, the schemes were adapted on the OpenCL framework and pixel shaders. The implementations were then evaluated using GPUs of the two biggest vendors. Considering OpenCL, the schemes exploit features of recent GPUs, such as warping. For CDF wavelets, the non-separable schemes exhibit a better performance than their separable counterparts on both the OpenCL and pixel shaders.

In the evaluation, we reached the following conclusions. Fusing several consecutive steps of the schemes might significantly speed up the execution, irrespective of their higher complexity. The non-separable schemes outperform their separable counterparts on numerous setups, especially considering the pixel shaders. All of the schemes are general and they can be used on any discrete wavelet transform. In future work, we plan to focus on general-purpose processors and multi-scale transforms.

Acknowledgements

This work has been supported by the Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science (no. LQ1602), and the Technology Agency of the Czech Republic (TA CR) Competence Centres project V3C – Visual Computing Competence Center (no. TE01020415).

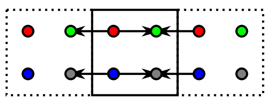
APPENDIX

For readers who are not familiar with signal-processing notations, a relationship between polyphase matrices and data-flow diagrams is explained here. The 2-D discrete wavelet transform divides the image into four polyphase components. Therefore, the 4×4 matrices of Laurent polynomials are used to describe the 2-D discrete wavelet transform. These matrices are commonly referred to as the polyphase matrices. The Laurent polynomials correspond to 2-D FIR filters, that define the transform. In most cases, the transform is described using a sequence of such matrices. One particular matrix thus defines a step of calculation in this case.

For example, the matrix

$$\mathrm{T}_{\mathrm{P}}^{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \mathrm{P} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \mathrm{P} & 1 \end{bmatrix}$$

maps four polyphase components to another four components, while using two 2-D FIR filters represented by the polynomials P. Moreover, when we substitute a particular polynomial, say $P(z) = -1/2(1+z^{-1})$, into the matrix, the mapping gets a specific shape. Such a substitution illustrated by the data-flow diagram in Figure 10. The solid arrows correspond to multiplication by $-1/2$ along with subsequent summation.



(a) $\mathrm{T}_{\mathrm{P}}^{H}$

Figure 10: Visual representation of the polyphase matrix. The four polyphase components are represented by color circles.

REFERENCES

[1] Barina, D., Kula, M., and Zemcik, P. Parallel wavelet schemes for images. *Journal of Real-Time Image Processing*, in press. doi: 10.1007/s11554-016-0646-3.

[2] Blazewicz, M., Ciznicki, M., Kopta, P., Kurowski, K., and Lichocki, P. *Two-Dimensional Discrete Wavelet Transform on Large Images for Hybrid Computing Architectures: GPU and CELL*, pages 481–490. Springer, 2012. ISBN 978-3-642-29737-3. doi: 10.1007/978-3-642-29737-3_53.

[3] Cohen, A., Daubechies, I., and Feauveau, J.-C. Biorthogonal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 45(5):485–560, 1992. ISSN 1097-0312. doi: 10.1002/cpa.3160450502.

[4] Daubechies, I. and Sweldens, W. Factoring wavelet transforms into lifting steps. *Journal of Fourier Analysis and Applications*, 4(3):247–269, 1998. ISSN 1069-5869. doi: 10.1007/BF02476026.

[5] Galiano, V., Lopez, O., Malumbres, M. P., and Migallon, H. Improving the discrete wavelet transform computation from multicore to GPU-based algorithms. In *Int. Conf. on Computational and Mathematical Methods in Science and Engineering*, pages 544–555, June 2011. ISBN 978-84-614-6167-7.

[6] Galiano, V., Lopez, O., Malumbres, M., and Migallon, H. Parallel strategies for 2D discrete wavelet transform in shared memory systems and GPUs. *The Journal of Supercomputing*, 64(1): 4–16, 2013. ISSN 0920-8542. doi: 10.1007/s11227-012-0750-5.

[7] Hsia, C. H., Guo, J. M., Chiang, J. S., and Lin, C. H. A novel fast algorithm based on smdwt for visual processing applications. In *IEEE International Symposium on Circuits and Systems*, pages 762–765, May 2009. doi: 10.1109/ISCAS.2009.5117860.

[8] Kucis, M., Barina, D., Kula, M., and Zemcik, P. 2-D discrete wavelet transform using GPU. In *International Symposium on Computer Architecture and High Performance Computing Workshop*, pages 1–6. IEEE, Oct. 2014. ISBN 978-1-4799-7014-8. doi: 10.1109/SBAC-PADW.2014.13.

[9] Kula, M., Barina, D., and Zemcik, P. New non-separable lifting scheme for images. In *IEEE International Conference on Signal and Image Processing*, pages 292–295. IEEE, 2016. ISBN 978-1-5090-2375-2. doi: 10.1109/SIPROCESS.2016.7888270.

[10] Mallat, S. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989. ISSN 0162-8828. doi: 10.1109/34.192463.

[11] Matela, J. GPU-based DWT acceleration for JPEG2000. In *Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, pages 136–143, 2009. ISBN 978-80-87342-04-6.

[12] Quan, T. M. and Jeong, W.-K. A fast discrete wavelet transform using hybrid parallelism on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, 27(11):3088–3100, Nov. 2016. ISSN 1045-9219. doi: 10.1109/TPDS.2016.2536028.

[13] Song, C., Li, Y., Guo, J., and Lei, J. Block-based two-dimensional wavelet transform running on graphics processing unit. *IET Computers Digital Techniques*, 8(5):229–236, Sept. 2014. ISSN 1751-8601. doi: 10.1049/iet-cdt.2013.0141.

[14] Sweldens, W. The lifting scheme: A custom-design construction of biorthogonal wavelets. *Applied and Computational Harmonic Analysis*, 3 (2):186–200, 1996. ISSN 1063-5203. doi: 10.1006/acha.1996.0015.

[15] Tenllado, C., Setoain, J., Prieto, M., Pinuel, L., and 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. ISSN 1045-9219. doi: 10.1109/TPDS.2007.70716.

[16] van der Laan, W. J., Jalba, A. C., and Roerdink, J. B. T. M. Accelerating wavelet lifting on graphics hardware using CUDA. *IEEE Transactions on Parallel and Distributed Systems*, 22(1):132–146, Jan. 2011. ISSN 1045-9219. doi: 10.1109/TPDS.2010.143.