Highly Effective GPU Realization of Discrete Wavelet Transform for Big-Data Problems

Dariusz Puchala[1][0000−0001−9070−8042]
and Kamil Stokfiszewski[1][0000−0002−2707−7353]

[1]Institute of Information Technology, Lodz University of Technology
ul. Wolczanska 215, 90-924 Lodz, Poland
{dariusz.puchala,kamil.stokfiszewski}@p.lodz.pl

Abstract. Discrete wavelet transform (DWT) is widely used in the tasks of signal processing, analysis and recognition. Moreover it's practical applications are not limited to the case of one-dimensional signals but also apply to images and multidimensional data. From the moment of introduction of the dedicated libraries that enable to use graphics processing units (GPUs) for mass-parallel general purpose calculations the development of effective GPU based implementations of one-dimensional DWT is an important field of scientific research. It is also important because with use of one-dimensional procedure we can calculate DWT in multidimensional case if only the transform's separability is assumed. In this paper the authors propose a novel approach to calculation of one-dimensional DWT based on lattice structure which takes advantage of shared memory and registers in order to implement necessary inter-thread communication. The experimental analysis reveals high time-effectiveness of the proposed approach which can be even 5 times higher for Maxwell architecture, and up to 2 times for Turing family GPU cards, than the one characteristic for the convolution based approach in computational tasks that can be classified as big-data problems.

Keywords: Discrete wavelet transform · GPU computations · Mass-parallel computations · Lattice structure · Time effectiveness.

1 Introduction

Discrete wavelet transform (DWT) is one of the basic tools of digital signal processing. It finds wide applications in such areas as processing of multidimensional data [1]-[2], image compression [3]-[4], image watermarking [5]-[6], analysis and clustering of high dimensional data [7, 8], data mining [9] and many more (see [10, 11]). It should be noted that even in case of images (or multidimensional data) one-dimensional wavelet transform procedure is a basic tool used in calculations due to the fact that two-dimensional (or respectively many dimensional) DWT can be calculated in a row-column scheme with use of one-dimensional transform. For this reason it is very important to develop time-effective realizations of one-dimensional DWT. In the recent years we could observe an intense research

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_19

on improvement of such algorithms (e.g. see [12]). This trend is especially visible thanks to the growing popularity of graphics processing units (GPUs) for which massively parallel algorithms for calculation of DWT were also constructed (c.f. [13]-[21]).

The solutions found in the literature are focused mainly on the case of two-dimensional data. For example in paper [18] the authors proposed the approach to calculation of 2D DWT based on row-column approach and the lifting scheme (see [10]) which takes advantage of shared memory and registers. In paper [16] the problem of calculation of 2D and 3D DWT based on row-column approach (and its extension to three dimensions) using shared memory and registers is addressed wherein the authors put a great effort on the issues of effective memory access. Finally in paper [17] we can find an approach based on lifting scheme where row and column passes are merged into inseparable transform. In selected papers we can find simple solutions to calculation of 2D DWT based on Haar wavelets and row-column scheme (e.g. see papers [14] and [15]). The problem of calculation of one-dimensional DWT was widely addressed in papers [19]-[21]. The authors of these papers analyzed convolution based approach and the approach taking advantage of lattice structure with multiple calls of kernel functions (we refer to this approach as a naive lattice structure in the remaining part of the paper). To the best knowledge of the authors of this paper the problem of effective calculation of one-dimensional DWT (i.e. using shared memory and registers) is not solved.

In this paper we propose a novel approach to calculation of one-dimensional wavelet transform. The proposed approach is based on the lattice structure and requires only one call of the kernel function. It also takes advantage of the shared memory and registers for communication between threads. Thanks to the mentioned features it can be characterized by high time-effectiveness. The experimental analysis shows that the proposed approach can be even 5 times faster for Maxwell architecture GPUs (NVIDIA GTX960), and even 2 times for Turing architecture (NVIDIA RTX2060), than the approach based on the convolution. Such acceleration is achieved mainly in case of big-data problems which makes the proposed approach particularly useful in modern computational tasks.

2 Calculation of one-dimensional DWT

In practical applications discrete one-dimensional wavelet transform is implemented as a two-channel filter bank with the structure shown in Fig. 2 (see e.g. [22]). Here an input signal $X(z) = X_0(z^2) + z^{-1}X_1(z^2)$ (with $X_0(z)$ and $X_1(z)$ describing even/odd numbered samples of input data) is in the first place decimated at blocks ($\downarrow 2$), which gives $[X_1(z), X_0(z)]^T$ vector, and then the same vector enters the block of filters defined by polyphase matrix $E(z)$ of the form:

$$E(z) = \begin{bmatrix} H_0(z) & H_1(z) \\ G_0(z) & G_1(z) \end{bmatrix},$$

where $H_0(z)$, $G_0(z)$ and $H_1(z)$, $G_1(z)$ represent, respectively, even/odd numbered coefficients of impulse responses of the filters. Both filters $H(z)$ and $G(z)$,

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_19

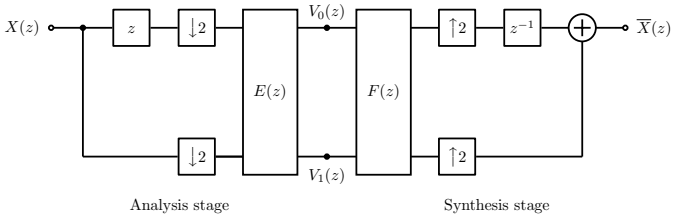

Fig. 1. The structure of two-stage filter bank in polyphase representation

together with decimators ($\downarrow 2$), form the analysis stage. At the output of this stage we get the signal in the form of $[V_0(z), V_1(z)]^T$ components, where $V_0(z)$ and $V_1(z)$ are obtained in the result of low-pass and high-pass filtering respectively (we refer here to the typical configuration of filters). The synthesis stage consists of filters described by the polyphase matrix $F(z)$. The synthesis and analysis filters are selected in a way satisfying the perfect reconstruction (PR) condition $F(z)E(z)=I$, with I being an identity matrix. With that assumption, and with use of upsampling blocks ($\uparrow 2$), it is possible to restore the input signal at the output of the bank, i.e. $\overline{X}(z) = X(z)$, if only there was no interference with the components obtained at the output of the analysis stage.

The practical implementations of the block of filters from Fig. 1 can be based on: (i) convolution approach, (ii) lattice structure, (iii) lifting scheme (see [10]). Lattice structure and lifting scheme approaches allow for almost two-fold reduction in the number of additions and multiplications and can be characterized by the smallest number of parameters required to describe both orthogonal and biorthogonal wavelet transforms, see e.g. [10]. This feature can be crucial from the point of view of adaptive parametric structures. In this paper we consider convolution approach, which is widely used in the literature (see [10]), and lattice structure approach, which was chosen as a basis for the proposed effective and mass-parallel solution. Morevoer, in the rest of the paper we assume N and M parameters to describe the size of input data and the length of the impulse responses of the filters respectively.

2.1 Calculations based on the convolution

The basic approach to calculation of DWT is based directly on the formula $[V_0(z), V_1(z)]^T = E(z)[X_1(z), X_0(z)]^T$. In practice it can be realized as the convolution of input signal with the impulse responses of both filters $H(z)$ and $G(z)$. Taking into account the polyphase representation of the bank and the decimation operations ($\downarrow 2$) it can be easily verified that the convolution should be calculated only at the position of even numbered samples of input signal. Such computational scheme offers good possibilities for mass parallel calculation of DWT. The approach adopted in this paper assumes the calculation of convolution in separate threads starting from each even numbered sample of input data. Moreover in order to reduce the number of memory transactions each thread realizes calculations for both filters $H(z)$ and $G(z)$ operating on the same data samples (see Fig. 2). This gives the number of $N/2$ threads required to process

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_19

the data. The results of computations must be written to the additional out-

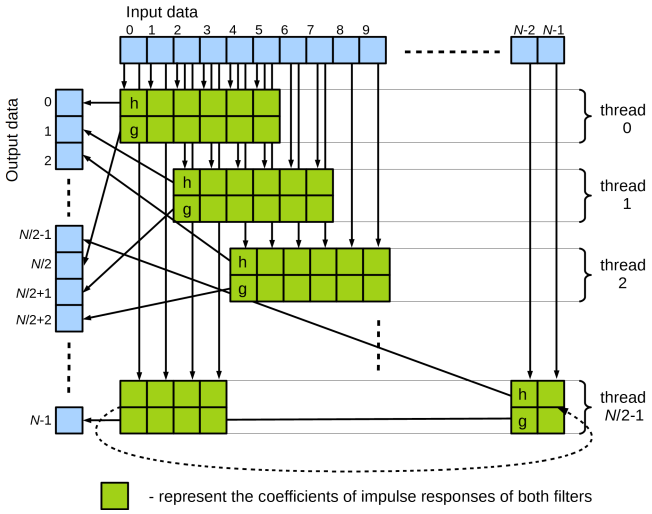

Fig. 2. Mass parallel computation of DWT based on one-dimensional convolution in case of periodic signals ($M=6$).

put table. Moreover all the calculations can be done fully independently, which means that no additional synchronization mechanism is required. As a result the implementation of mass-parallel convolution based calculations of DWT requires one execution of the kernel function. It should be noted that Fig. 2 shows the case where the periodic repetition of input data is assumed. This is the boundary condition considered in this paper. The additional analysis of the computational structure reveals that it can be characterized by a number of approximately 4 arithmetical operations performed for a single data transaction. It is enough to consider operations within one thread, where we have: $4M-2$ arithmetical operations (additions and multiplications) and $M+2$ data transactions (data read and write operations). The total numbers of additions ($\mathcal{L}_{ADD}$) and multiplications ($\mathcal{L}_{MUL}$) required by the convolution approach can be described by the formulas:

$$\mathcal{L}_{ADD}=N(M-1), \quad \mathcal{L}_{MUL}=NM. \tag{1}$$

In turn the total number of memory transactions ($\mathcal{L}_{MEM}$), a number of sequential steps ($\mathcal{L}_{SEQ}$) (considers arithmetical operations when calculations within all threads can be realized at the same time) and a number of kernel function calls equal respectively:

$$\mathcal{L}_{MEM}=\frac{1}{2}N(M+2), \quad \mathcal{L}_{SEQ}=4M-2, \quad \mathcal{L}_{KER}=1. \tag{2}$$

It should be noted that the number of sequential steps that must be performed is a measure of complexity of parallel algorithm.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_19

2.2 Calculations based on the lattice structure

Although the computations within a filter bank are effective and can be described by linear complexity $\mathcal{O}(MN)$, where N is the size of input signal, and M describes the size of filters, its efficiency can be increased even more with use of lattice structures. Lattice structures allow for almost twofold reduction in the number of multiplications and additions and allow for accurate parametrization in the sense of a number of free parameters (see [10]). The construction of a lattice structure requires factorization of the polyphase matrix $E(z)$ into a product of simple matrices. Following [10] such factorization can be described as:

$$E(z) = BD\Lambda(z^{-1}) \left(\prod_{i=2}^{\frac{M}{2}-1} A_{\frac{M}{2}-i} D\Lambda(z^{-1}) \right) A_0, \tag{3}$$

where the matrices used in formula (2) can be defined as:

$$B = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad A_i = \begin{bmatrix} 1 & t_i \\ s_i & 1 \end{bmatrix}, \quad \Lambda(z) = \begin{bmatrix} 1 & 0 \\ 0 & z \end{bmatrix}, \quad D = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Matrices B and A_i for $i = 0, 1, \ldots, \frac{M}{2} - 2$ represent basic operations within the lattice structure which are parametrized with the values of a, b, c, d and t_i, s_i for $i = 0, 1, \ldots, \frac{M}{2} - 2$ parameters. It should be noted that in case of the orthogonal bank of filters the following relations take place: $s_i = -t_i$ for $i = 0, 1, \ldots, \frac{M}{2} - 2$ and $d = \pm a$, $c = \mp b$. For example orthogonal Daubechies 4 (db4) wavelet can be decomposed into the set of parameters: $t_0 = -0.322276$, $t_1 = -1.23315$, $t_2 = -3.856628$ and $a = 0.15031$, $b = 0.006914$ and $d = a$, $c = -b$. In case of biorthogonal bank of filters all of the parameters are required in general, though the well-known postulate of multi-resolution analysis, i.e. $H_0(-1) = 0$ and $H_1(1) = 0$ (c.f. [10]), which must be met by any wavelet transform, makes the values of b and c dependent on the values of the remaining parameters. Then the formula (3) describes the accurate factorization of matrix $E(z)$ where the number of free parameters is the smallest possible depending on the considered family of wavelet transforms. In Fig. 3 we can see the data flow diagram of the lattice structure based on the decomposition formula (3).

In case of the lattice structure parallelization of calculations is possible due to the independent character of computations realized within A_i matrices for $i = 0, 1, \ldots, M/2 - 2$ and B matrix at each stage of calculations (see Fig. 3). The operations described by A_i and B matrices are represented graphically as butterfly operators '$\bullet$' and '$\circ$' respectively. It should be noted that at each stage the operations within butterfly operations are independent, since each butterfly operator operates on an individual pair of input data. If we assume that each butterfly operator is implemented by one thread then the total required number of threads will be $N/2$. However, it is necessary to synchronize the calculations between consecutive stages which is possible with use of global synchronization mechanisms. Hence, the mass-parallel realization of the lattice structure based directly on the data flow diagram from Fig. 3 requires $M/2$ calls of the kernel

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_19

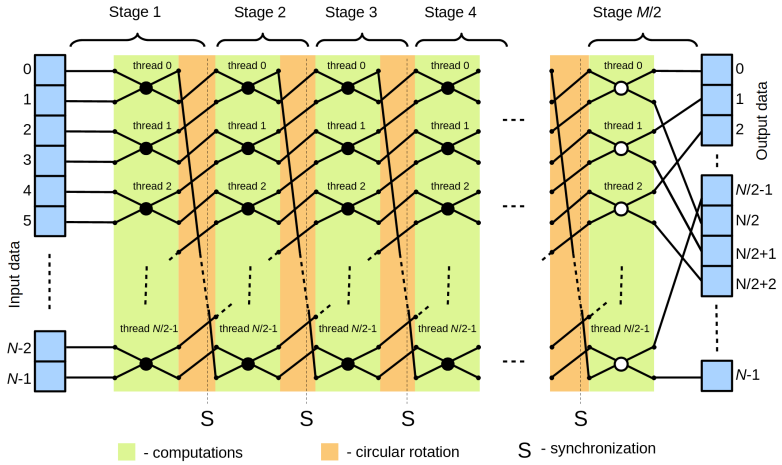

Fig. 3. Mass-parallel computation of DWT based on one-dimensional lattice structure in case of periodic signals ('•' - operations described by matrices A_i for $i = 0, 1, \ldots, M/2-2$; '∘' - operations described by matrix B).

function. Since the time required to call a kernel function is a thousand times longer than a time of a single mathematical operation we will call this approach naive. Further analysis allows to say that the lattice structure described by the factorization formula (3) can be characterized by a number of 1 (for A_i operators) and 3/2 (for B operators) arithmetical operations for one data transaction. The total numbers of additions and multiplications required by the lattice structure can be expressed respectively as:

$$\mathcal{L}_{ADD} = \frac{1}{2}NM, \quad \mathcal{L}_{MUL} = \frac{1}{2}N\,(M+2)\,. \tag{4}$$

Moreover the total number of memory transactions, sequential steps and the calls of kernel function can be described by:

$$\mathcal{L}_{MEM} = NM, \quad \mathcal{L}_{SEQ} = 2(M+1), \quad \mathcal{L}_{KER} = \frac{1}{2}M. \tag{5}$$

Summarizing, the direct comparison of both described approaches indicates approximately twofold reduction of the number of arithmetic operations in case of lattice structure which translates into twofold reduction of the sequential steps required by the mass parallel realizations of both approaches. However, the convolution based approach can be characterized by almost twice smaller number of memory transaction than the lattice structure. Moreover the naive lattice structure requires much higher number of kernel function calls. It should be noted that a single kernel function call may take even several thousands of clock cycles while a single mathematical operation requires around 20 cycles. Hence such an approach may be characterized by poor time-effectiveness.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_19

3 The proposed lattice structure

In this paper we propose a novel approach to calculation of DWT based on the lattice structure. We choose the lattice structure as the starting point since it can be characterized by almost two times smaller number of arithmetic operations than the convolution based approach. This feature can be crucial in practical scenarios when operating on big data sets because GPU computations have both sequential and parallel character. The main goal of introducing a new approach is to increase the efficiency of DWT calculations. The increase in efficiency is possible if we reduce a number of kernel function calls and take advantage of shared memory and effective local synchronization mechanism. The mentioned objectives can be achieved only if the computations within the entire lattice structure can be partitioned into separate and independent sets of operations which can be executed within separate blocks of threads. However, if we look at the structure from Fig. 3, it is difficult to distinguish separate groups of base operations due to the specific permutation (cyclic rotation) of data between successive stages. Such partitioning is possible, however, if we accept the possibility of repeating some basic operations. The proposed approach is shown in Fig. 4.

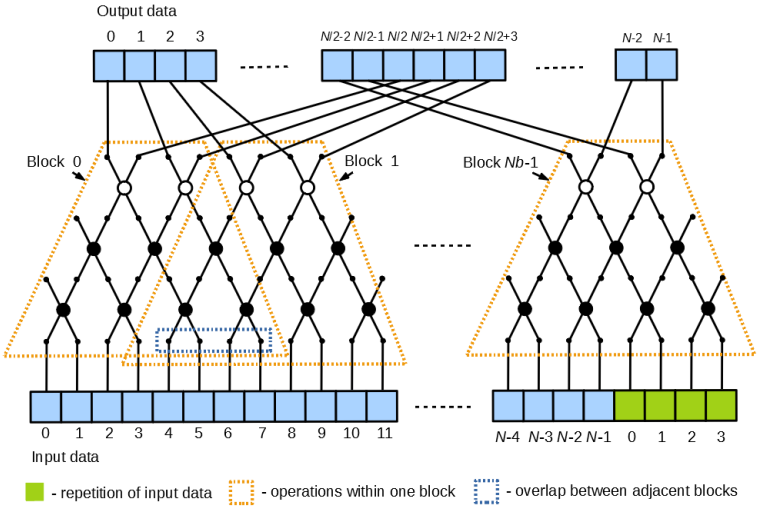

Fig. 4. The organization of calculations in case of the proposed approach to mass-parallel realization of DWT ($M = 6$).

In the proposed approach, the base operations are grouped into independent blocks according to the presented partition scheme (see Fig. 4). Each block contains the set of base operations from successive stages that create trapezoidal structures (indicated with orange dotted lines). Although adjacent blocks overlap by $K = M - 2$ elements of the input table, computations across all blocks are independent and may be performed at the same time. In addition, operations within a single block can take advantage of shared memory as well as efficient synchronization mechanism in the form of the `__syncthreads()` function. In Fig. 5 we show the organization of calculations within a single block.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_19

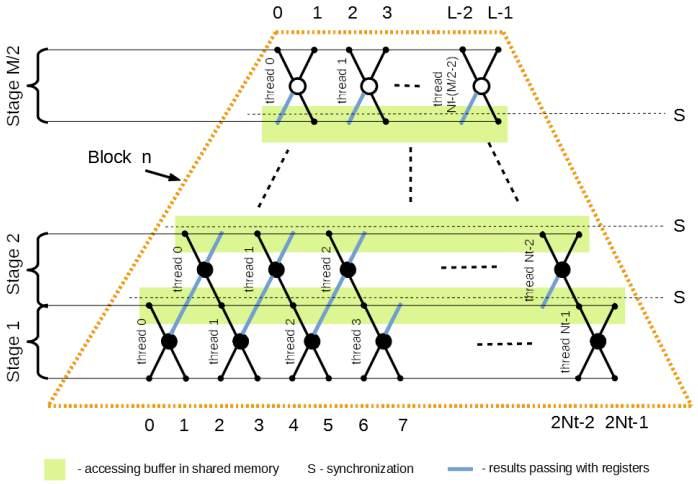

Fig. 5. Operations realized within one block in case of the proposed approach.

The size of input data at the input of the block equals $2N_t$, where N_t is a number of threads within a block. It should be noted that every thread in the block is responsible for realization of one butterfly operator, i.e. realizations of operations required by A_i for $i=0,1,\ldots,\frac{M}{2}-2$ and B matrices. However, in the following stages, the number of threads used decreases by one with each stage. The final number of active threads depends on the filter size and equals $N_t-\frac{M}{2}-1$. Hence, the size of output data for a single block equals $L=2N_t-K$ with $K=M-2$. The role of the base operations in the first stage is to read data from the global memory, perform appropriate operations and write the results to registers or shared memory. Since the same threads realize operations in the following stages, it becomes possible to pass the results directly through the registers (blue connections between butterfly operations in Fig. 5). In this way, the base operations within internal stages read data from registers or shared memory and write the results of calculations at the same localizations. The operations at the last stage write the results to the global memory. Synchronization between stages is realized with use of the fast intra-block synchronization.

It should be noted that with such organization of calculations the realization of the assumed boundary condition requires to extend the input table by $M-2$ elements and fill them with the first $M-2$ elements of the input data (see the green elements of input table in Fig. 4). In this way the size of input table for the whole structure from Fig. 4 equals $N+(M-2)$, where N is the size of input data. The size of input data, which is at the same time the size of the output of the whole structure, can be calculated as $N=N_bL$, where N_b is the number of blocks of threads. The further analysis of the proposed approach allows to derive the formulas for the number of additions ($\mathcal{L}_{ADD}$) and multiplications ($\mathcal{L}_{MUL}$):

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_19

$$\mathcal{L}_{ADD}=\frac{1}{2}NM+\left(\frac{1}{4}N_bMK\right),\quad \mathcal{L}_{MUL}=\frac{1}{2}N(M+2)+\left(\frac{1}{2}N_bK\left(\frac{1}{2}K+1\right)\right), \quad (6)$$

and also the numbers of kernel calls ($\mathcal{L}_{KER}$) and the sequential steps ($\mathcal{L}_{SEQ}$) that must be realized within the algorithm:

$$\mathcal{L}_{KER}=1,\quad \mathcal{L}_{SEQ}=2(M+1) \quad (7)$$

The comparison of formulas (4) and (6) allows to state that the proposed approach can be characterized by the higher number of arithmetical operations resulting from the overlaps between the blocks of threads. However, the comparison of formulas (5) and (7) allows to see that the number of sequential steps stays the same but the number of kernel function calls in the proposed approach equals 1. Also the analysis of the number of global memory transactions, where in case of the proposed approach we have:

$$\mathcal{L}_{GMEM}=2N+N_bK,\quad \mathcal{L}_{SMEM}=\frac{1}{2}N_bK\left(2N_t-\frac{1}{2}K\right)$$

with $\mathcal{L}_{GMEM}$ and $\mathcal{L}_{SMEM}$ describing the numbers of global and shared memory transactions respectively, allows to state that the proposed approach reduces significantly the number of global memory transactions for shared memory transactions and register based data passing.

In the proposed implementation (see Listing 2) the kernel function `proposed()` is called only once. The arguments of the kernel function include `input[]` and `output[]` buffers intended for input data and the results of computations respectively, as well as the `coeff[]` table holding the parameters required by the operators at the following stages of the lattice structure. The remaining three parameters are: `N2` $=\frac{N}{2}$, `L` – the filter length and `m2` $=\frac{M}{2}$. It should be noted that buffer `sbuf[]` of size N_t elements is located in the shared memory and also the passing of the intermediate results of computations between stages through registers is realized with use of `fVal[]` variable.

In order to avoid a situation in which the elements of input data are first read by the threads operating on even indexes and next operating on odd indexes, thus making the memory references not optimally organized, we improve the efficiency of memory transfers by introducing `ll_to_ff()` function (see Listing 1), which reads two `float` elements as one 64-bit reference to `long long` type. Then using dedicated CUDA API function and bit-shift operation it is possible to extract two 32-bit `float` variables from the value of one 64-bit variable.

```
__device__ inline void ll_to_ff(long long* d,float& a, float& b) {
    long long bb;
    bb=*((long long*)d);
    a=__uint_as_float((unsigned int)bb);
    b=__uint_as_float((unsigned int)(bb>>32));
}
```

Listing 1. Function reading two `float` variables as one 64-bit reference.

Using the function `ll_to_ff()` form Listing 1 we come up with the following kernel code implementing the proposed DWT computation method.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_19

```
__global__ void proposed(float* input,float* output,
                         float* coeff,int N2,int L,int m2) {
    __shared__ float sbuf[Nt];   // Shared memory buffer
    int iInd0,iInd1,iInd2,iInd3; // Additional variables in registers
    float a,b,c,d,fX,fY,fVal;
    // Initialization and mapping of thread coordinates to data index
    iInd0=1;
    iInd1=threadIdx.x;
    iInd2=L2*blockIdx.x+threadIdx.x;
    iInd3=blockDim.x-1;
    // First stage: reading input data and values of operation parameters
    ll_to_ff((long long*)coeff,a,b);
    ll_to_ff((long long*)input+iInd2,fX,fY);
    // Computations within the first stage
    sbuf[iInd1]=fX+a*fY;
    fVal=b*fX+fY;
    __syncthreads();  // Synchronization
    while(iInd0<m2) { // Internal stages
        iInd1+=1;
        if (threadIdx.x<iInd3) { // Filtering of unneeded threads
            // Reading operation parameters
            ll_to_ff((long long*)coeff+iInd0,a,b);
            // Computations within stage
            fY=sbuf[iInd1];
            sbuf[iInd1]=fVal+a*fY;
            fVal=b*fVal+fY;
        }
        iInd0+=1;
        iInd3-=1;
        __syncthreads();     // Synchronization
    }
    if (threadIdx.x<iInd3) { // Last stage
        // Reading operator parameters
        iInd1+=1;
        ll_to_ff((long long*)coeff+iInd0,a,b);
        iInd0+=1;
        ll_to_ff((long long*)coeff+iInd0,c,d);
        // Computations within stage and storing results to global memory
        fY=sbuf[iInd1];
        output[iInd2]=a*fVal+b*fY;
        output[iInd2+N2]=c*fVal+d*fY;
    }
}
```

Listing 2. Kernel function for the proposed approach.

4 Experimental analysis

In order to verify the effectiveness of the proposed approach a series of experiments was performed including various sizes of filters and input data. During the experiments we considered filter sizes M changing in a range from 4 to 32 coefficients and the experimental data of sizes between 1024 and $1024 \cdot 10^5$ elements. In Fig. 6 we present selected experimental results obtained with NVIDIA GTX960 GPU card (Maxwell family) which is characterized by the total number of 1024 computing cores and 4GB of global memory (128-bit memory bus and 112.2GB/s of memory bandwidth). The second GPU card used was NVIDIA RTX2060. It is a representative of the Turing architecture of NVIDIA GPU cards with a number of 1920 computing cores and 6GB of global memory (192-bit memory bus and 336GB/s of memory bandwidth). The selected results obtained for the second card are presented in Fig. 6.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_19

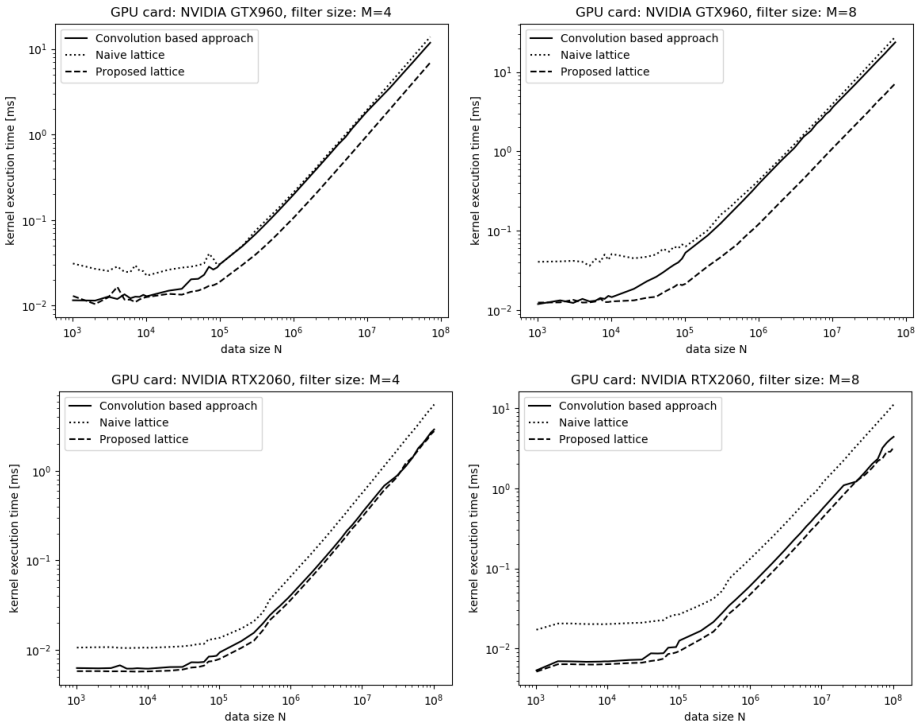

Fig. 6. Comparison of time of calculations of DWT for NVIDIA GTX960 and NVIDIA RTX2060 cards and filter sizes $M=4$ and $M=8$.

In Fig. 6 we present the results obtained with NVIDIA GTX960 card for the popular filter sizes, i.e. $M \in \{4, 8\}$. The analysis of results indicates a huge advantage of the convolution approach over the approach using the naive lattice for data sizes up to around 10^5 elements. The ratio of execution times (convolution time/lattice time)[1] lies between 0.15 and 1.0 depending on the filter and input data sizes. For example for $M=4$ the ratio is below 0.50 for smaller data sizes ($N \leq 0.5*10^5$), and grows up to 1 at $N \approx 10^5$ to stay close to 0.9 for the remaining data sizes. It shows that the naive lattice can be more than 2 times slower than the convolution based approach. For $M=8$ the ratio changes between 0.3 and 0.93 depending on the size of input data. The most favorable results for the naive lattice are obtained with filters of size $M=32$ and indicate only 15% advantage of the convolution approach for data sizes $N \geq 0.25 * 10^5$. On this basis we state that naive lattice allows to obtain much worse results than the results possible to obtain with use of convolution. Hence, the naive lattice is characterized by significantly lower time-efficiency than the convolution approach.

In case of the proposed approach the situation looks differently. For filter sizes $M \in \{4, 8\}$ (highly practical sizes of filters), and starting from data size

[1] We measured times of kernel launch and calculations with NVIDA's `nvprof` profiler. During experiments we used Intel i7-9700, 12 MB cache, 32 GB RAM, Windows 10 platform, and CUDA 10, C++ implementations.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_19

of around 10^4 elements, the proposed approach can be characterized by much higher effectiveness than the convolution based approach. For the considered filter sizes the obtained ratios of times of calculations are around 1.9, 2.7 and 3.4 respectively. Even for sizes smaller than 10^4 the ratio changes between 0.9 and 1.1. In Fig. 7(a) we present the values of ratios of computation times between the convolution and the proposed approach for all of the considered sizes of filters and input data. In case of NVIDIA GTX960 card the data size of around 10^4 is the lower limit of effectiveness of application of the proposed approach. It should be noted, however, that the highest advantage of the proposed approach over the convolution one is above 5 times and is obtained for high filter sizes ($M \geq 24$).

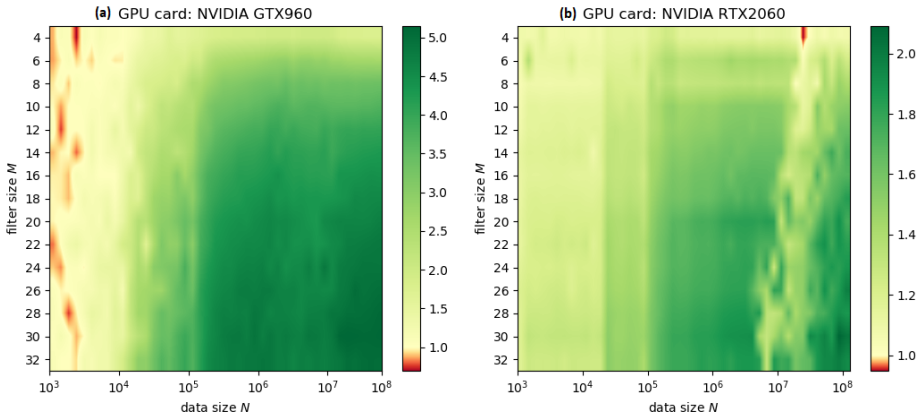

Fig. 7. Ratios of times of calculations (convolution time/proposed lattice time) of DWT for NVIDIA's (a) GTX960 and (b) RTX2060 cards.

In case of NVIDIA RTX2060 card, see Fig. 6, the naive lattice obtains significantly worse results. Here the ratios are in a range 0.1 to 0.3. So the convolution approach is in the worst case even 10 times faster. The proposed approach allows to obtain better results than the convolution for all data sizes and all sizes of filters (there are a few exceptions for $M \in \{4, 8\}$ and $N \approx 10^8$). For small filter sizes, like $M \in \{4, 8\}$, the proposed approach is up to 1.25 and 1.5 times faster respectively. The highest advantage can be observed for long filters $M \in \{30, 32\}$ and high sizes of input data and it equals even 2 times, see Fig. 7(b). Moreover, if we compare results of the proposed approach obtained with both considered GPUs, then the possible speed-up of computations obtained with NVIDIA RTX2060 is in the range $[1.75, 4.12]$, while the average value is around 2.68 times. The averaged practical performance of the proposed method is 83 GFlops/s for NVIDIA GTX960, and 233 GFlops/s for NVIDIA RTX2060.

The results of the proposed approach for NVIDIA RTX2060 card where compared to sequential convolution and lattice CPU implementations. The CPU variants allowed to obtain better results for small amount of processed data, i.e. $MN \leq 16384$ for convolution, or $MN \leq 32768$ for lattice approach. Here, the computation time ratios between CPU and GPU were within the range $[0.20, 0.99]$, e.g. for lattice approach with $M = 4$, and $N = 1024$, the CPU implementation

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_19

was 5 times faster than the proposed approach for GPU. However, for longer filters and input data sizes the maximum speed-up of computations obtained with the proposed approach was more than 200 or 120 times, respectively, when compared to convolution and lattice based CPU implementations.

On the basis of the conducted experimental analysis we can state that the proposed approach to calculation of one-dimensional wavelet transform allows to obtain better results than CPU lattice approach, and the popular and widely used convolution approach for GPU. The possible advantage is several hundred times over CPU, and several times in case of GPU, and it can be observed for large data sets. Thus, we conclude that the proposed approach is highly effective for big-data problems.

5 Conclusions

In this paper the authors propose a novel approach to calculation of one-dimensional discrete wavelet transform. The presented solution takes advantage of shared memory and registers to implement communication between threads. Moreover only one call of a kernel function is required. The computations within the proposed approach are based on the lattice structure which is an effective tool for calculation of one-dimensional wavelet transform. The experimental analysis involving two models of GPU cards (NVIDIA GTX960 and NVIDIA RTX2060) reveals high time-effectiveness of the proposed solution. It is significantly faster than popular solutions dedicated for GPUs and known from the literature, i.e. the convolution based approach and the approach based on naive lattice structure that requires multiple calls of kernel functions (see papers [19]-[21]). The experimental analysis shows that the obtained acceleration can be even 3 times (for NVIDIA GTX960) and 1.5 times (for NVIDIA RTX2060) for practical sized of filters ($M \in \{4, 6, 8\}$) when compared to the convolution based approach which highly outperforms the solution based on naive lattice structure. The highest acceleration is obtained with long filters (i.e. $M = 32$) and long sizes of input data and it equals even 5 times in case of NVIDIA GTX960 card (and up to 2 times for NVIDIA RTX2060). On the basis of the experimental analysis we conclude that the proposed approach fulfills its role and can find practical applications with particular emphasis on big-data problems.

References

1. A. Nakonechny, Z. Veres, *The Wavelet Based Trained Filter for Image Interpolation*, Proc. of IEEE First International Conference on Data Stream Mining & Processing, pp. 218-221, 2016.
2. A. Galletti, L. Marcellino, L. Russo, *A GPU Parallel Algorithm for Image Denoising Based on Wavelet Transform Coefficients Thresholding*, 14th Int. Conf. on Signal-Image Technology & Internet-Based Systems (SITIS), pp. 485-492, 2018.
3. A. A. Nashat, N. M. Hussain Hassan, *Image compression based upon Wavelet Transform and a statistical threshold*, International Conference on Optoelectronics and Image Processing (ICOIP), pp. 20-24, 2016.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_19

4. A. Paul, T. Z. Khan, P. Podder, R. Ahmed, M. M. Rahman, M. H. Khan, *Iris image compression using wavelets transform coding*, 2nd International Conference on Signal Processing and Integrated Networks (SPIN), pp. 544-548, 2015.
5. P. Lipiński, J. Stolarek, *Improving Watermark Resistance against Removal Attacks Using Orthogonal Wavelet Adaptation*, Proc. 38th Conf. on Current Trends in Theory and Practice of Computer Science Location, Vol. 7147, pp. 588-599, 2012.
6. J. Liu, *An Image Watermarking Algorithm Based on Energy Scheme in the Wavelet Transform Domain*, IEEE 3rd International Conference on Image, Vision and Computing (ICIVC), pp. 668-672, 2018.
7. S. Babichev, *Optimization of Information Preprocessing in Clustering Systems of High Dimension Data*, Radio Electr., Comp. Sci., Control, No. 2, pp. 135-142, 2014.
8. B. Kim, T. McMurry, W. Zhao, R. Wu, A. Berg, *Wavelet-Based Functional Clustering for Patterns of High-Dimensional Dynamic Gene Expression*, Journal of Computational Biology, Vol. 17, No. 8, pp. 1067-1080, 2010.
9. J. Dong, *Data Mining of Time Series Based on Wave Cluster*, International Forum on Information Technology and Applications, IFITA '09, 2009.
10. T. Nguyen , G. Strang, *Wavelets and Filter Banks*, Cambr. Press, Welleslay, 1996.
11. S. G. Mallat, *A theory for multiresolution signal decomposition: the wavelet representation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 11, No. 7, pp. 674-693, 1989.
12. M. Yatsymirskyy, *Lattice structures for synthesis and implementation of wavelet transforms*, Jour. of Applied Computer Science, Vol. 17, No. 1, pp. 133-141, 2009.
13. H. Sorensen, *High-Performance Matrix-Vector Multiplication on the GPU*, M. Alexander et al. (Eds) Euro-Par 2011 Parallel Processing Workshops, Lecture Notes in Computer Science, Vol. 7155. Springer, Berlin, Heidelberg, 2012.
14. E. T. Angaji, S. A. R. Ebrahimi, *Accelerating Haar wavelet transform with CUDA-GPU (July 2017)*, 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), pp. 791-796, 2017.
15. S. Aslan, H. Badem, D. Karaboga, A. Basturk, T. Ozcan, *Accelerating Discrete Haar Wavelet Transform on GPU cluster*, 9th International Conference on Electrical and Electronics Engineering (ELECO), pp. 1237-1240, 2015.
16. T. M. Quan, W. Jeong, *A Fast Discrete Wavelet Transform Using Hybrid Parallelism on GPUs*, in IEEE Transactions on Parallel and Distributed Systems, Vol. 27, No. 11, pp. 3088-3100, 2016.
17. D. Barina, M. Kula, M. Matysek, P. Zemcik, *Accelerating discrete wavelet transforms on GPUS*, IEEE Int. Conf. Image Processing (ICIP), pp. 2707-2710, 2017.
18. P. Enfedaque, F. Auli-Llinas, J. C. Moure, *Implementation of the DWT in a GPU through a Register-based Strategy*, in IEEE Transactions on Parallel and Distributed Systems, Vol. 26, No. 12, pp. 3394-3406, 2015.
19. D. Puchala, B. Szczepaniak, M. Yatsymirskyy, *Lattice structure for parallel calculation of orthogonal wavelet transform on GPUs with CUDA architecture*, Przeglad Elektrotechniczny, Vol. R.91, No. 7, pp. 52-54, 2015.
20. D. Puchala, K. Stokfiszewski, B. Szczepaniak, M. Yatsymirskyy, *Effectiveness of Fast Fourier Transform implementations on GPU and CPU*, Przeglad Elektrotechniczny, Vol. 92, No. 7, pp. 69-71, 2016.
21. D. Puchala, K. Stokfiszewski, K. Wieloch, M. Yatsymirskyy, *Comparative Study of Massively Parallel GPU Realizations of Wavelet Transform Computation With Lattice Structure And Matrix-Based Approach*, IEEE Second International Conference on Data Stream Mining & Processing (DSMP), pp. 88-93, 2018.
22. T. Cooklev, *An efficient architecture for orthogonal wavelet transforms*, IEEE Signal Processing Letters, Vol. 13, No. 2, pp. 77-79, 2006.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_19