Wavelet Lifting on Application Specific Vector Processor

David Barina Pavel Zemcik

Faculty of Information Technology
Brno University of Technology
{ibarina, zemcik}@fit.vutbr.cz

Abstract

With the start of the widespread use of discrete wavelet transform the need for its efficient implementation is becoming increasingly more important. This work presents a general approach of discrete wavelet transform scheme vectorisation evaluated on an FPGA-based Application-Specific Vector Processor (ASVP). This unit can be classified as SIMD computer in Flynn's taxonomy. The presented approach is compared with two other non-vectorised approaches. Using the frequently exploited CDF 9/7 wavelet, the achieved speedup is about $2.6\times$ compared to naive implementation.

Keywords: discrete wavelet transform, lifting scheme, SIMD, parallelization, vectorisation

1 Introduction

The discrete wavelet transform (DWT) is mathematical tool which is able to decompose discrete signal into lowpass and highpass frequency components. Such a decomposition can be performed at several scales. DWT is often used as the basis of sophisticated compression algorithms. This is the case of JPEG 2000 and Dirac compression standards in which CDF 9/7 wavelet [5] is employed for lossy compression. Responses of this wavelet can be computed by a convolution with two FIR filters, one with 7 and the other with 9 coefficients. For the DWT computation, the well known Mallat's [9] filtering scheme can be used. Alternatively, one can use usually faster scheme called lifting which was presented by I. Daubechies and W. Sweldens in [6]. Lifting data flow graph consists of regular grid computational scheme suitable for SIMD vectorisation. Both of the algorithms can be performed over some approximation of real numbers. This paper focuses on single-precision floating-point (SP FP) format.

In the field of FPGA-based accelerators, the platform called Application-Specific Vector Processor (ASVP) was recently proposed [11], [10]. This platform uses several simple units referred as Basic Computing Elements (BCEs). BCEs are able to accelerate simple operations (like addition or multiplication) on long single-precision floating-point vectors. Thus, these units can be classified as SIMD computers in Flynn's taxonomy. Lifting of CDF 9/7 transform can be directly adapted on them.

In this work, we discuss vectorisation (parallelization) of discrete wavelet transform on ASVP platform. The rest of the paper is organized as follows. More traditional approaches to DWT computation together with ASVP platform are reviewed in Section 2. Section 3 gives the basics behind the lifting scheme. Section 4 describes opportunities for lifting scheme parallelizations and presents our proposed approach. Introduced parallelization methods are compared in section 5. Finally, Section 6 concludes the paper.

2 Related work

In 2000, the problem of minimum memory implementations of lifting scheme was addressed in [4] by Ch. Chrysafis and A. Ortega. This approach is very general and it is not focused on parallel processing. The work was also later extended to [3] where same authors addressed a problem of minimum memory implementation of 2-D transform. Also, variation of this approach was presented six year later in [7] which is specifically focused on CDF 9/7 wavelet transform.

In [8] R. Kutil *et al.* presented SIMD vectorisation of several frequently used wavelet filters. This vectorisation is applicable only on those filters discussed in their paper. Specifically, vectorisation of CDF 9/7 wavelet computed using lifting scheme is vectorised here by a group of four successive pairs of coefficients. For CDF 9/7 wavelet discussed in this paper, their measurements gave a speedup of 2.65 for forward and 1.7 for backward transform on Intel Pentium 4 with SSE extension.

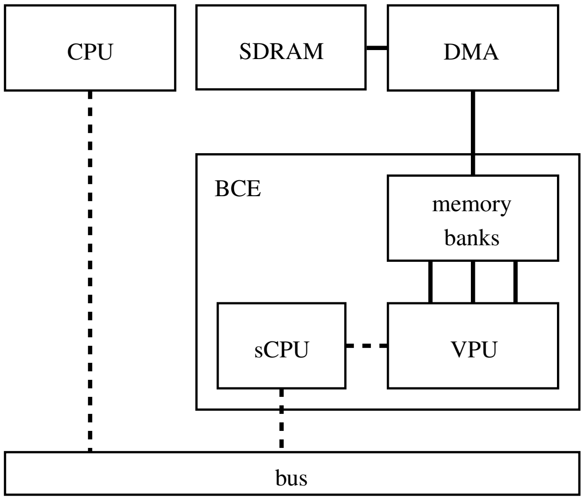

Figure 1: *Organization of ASVP platform. Solid lines indicates data paths. In our case, function of BCE is controlled by host CPU (MicroBlaze). The sCPU means for simple CPU what is the PicoBlaze processor here. Moreover, the BCE element consists of four memory banks, each 1024 32-bit words long, and Vector Processing Unit (VPU) which performs actual operations. The BCE accesses RAM through DMA engine.*

The platform used in this paper is Application-Specific Vector Processor (ASVP, originally EdkDSP) recently presented in [11], [10], [1] and [2]. This heterogeneous multi-core platform employs up to several units called Basic Computing Element (BCE) which can accelerate floating-point vector operations. For organization of ASVP see Figure 1. These elements use a combination of a simple PicoBlaze CPU (sCPU in Figure 1) with a configurable pipelined datapath. The computation performed by BCE can be changed through replacing the PicoBlaze firmware. Moreover, the ASVP platform contains host CPU (MicroBlaze in this case) that is executing the main program. Thus, the computation is distributed between host CPU and one or more BCE units. This change of the BCE firmware can be made from MicroBlaze CPU in runtime. The BCE contains four memory banks each of 1024 words long (one word denotes 32-bit SP FP). Before BCE can start its program, the input data must be transferred from main DDR memory into BCE's memory banks. Similarly, the output data should be transferred back when BCE computation is done. These data are transferred by DMA controller. The operations performed by BCE are element-wise move, addition, multiplication, etc.

3 Lifting scheme

According to the number of arithmetic operations, the lifting scheme [6] is today's most efficient scheme for computing discrete wavelet transforms. Any discrete wavelet transform with finite filters can be factored into a finite sequence of N pairs of predict and update convolution operators P_n and U_n. Each predict operator P_n corresponds to a filter $p_i^{(n)}$ and each update operator U_n to a filter $u_i^{(n)}$.

$$P_n(z) = \sum_{i=-l_n}^{g_n} p_i^{(n)} z^{-i} \tag{1}$$

$$U_n(z) = \sum_{i=-m_n}^{f_n} u_i^{(n)} z^{-i} \tag{2}$$

This factorisation is not unique. For symmetric filters, this non-uniqueness can be exploited to maintain symmetry of lifting steps.

Consider the decomposition of the signal of length of L samples. Without loss of generality one can assume only signals with even length L. Possible remaining coefficient can treated separately in the prolog or epilog phases together with border extension. Thus, the transform contains $S = L/2$ pairs of resulting wavelet coefficients (s, d). The s coefficients represent a smoothed signal. On the contrary, the d coefficients form a difference or detail signal.

In their paper [6], Daubechies and Sweldens demonstrated an example of CDF 9/7 transform factorisation which resulted into four lifting steps ($N = 2$) plus scaling of coefficients. In this example, the individual lifting steps use 2-tap symmetric filters for the prediction as well as the update. In all figures shown in this paper, the coefficients of these four 2-tap symmetric filter are denoted α, β, γ and δ respectively.

When coefficient scaling is omitted, the calculation of a pair of the DWT coefficients at the position l (s_l and d_l) is performed through four lifting steps. Intermediate results ($s_l^{(n)}$ and $d_l^{(n)}$) can be appropriately shared between neighbouring pairs of coefficients (s_l and d_l). Finally, the calculation of the complete CDF 9/7 DWT is depicted in Figure 2. This is an in-place implementation, which means the DWT can be calculated without allocating auxiliary memory. Resulting coefficients (s_l and d_l) are interleaved in place of the input signal.

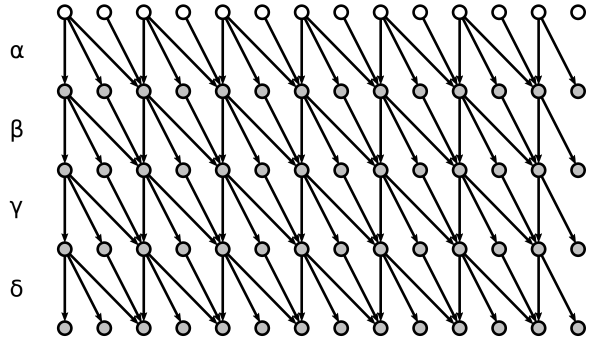

Figure 2: *Complete data flow graph of CDF 9/7 wavelet transform. The input signal is on top, output at the bottom. The graph borders must be treated in a special way using prolog and epilog phases.*

4 Vectorisation

The calculation scheme described in the previous section can be realized in a number of different ways. In this work, two of such ways are described. The main difference between them is in the order of lifting steps evaluation. Alternatively, the data flow graph in Figure 2 can be split into areas that are evaluated sequentially according to their data dependencies.

4.1 Horizontal vectorisation

The naive approach of data flow graph evaluation directly follows the lifting steps (n). Thus, all intermediate $s^{(1)}$ and $d^{(1)}$ coefficients are evaluated in the first step. Then, all $s^{(2)}$ and $d^{(2)}$ are evaluated in second step, etc. Unfortunately, this algorithm requires several reads and writes of the intermediate results $s_l^{(n)}$ and $d_l^{(n)}$. For long signals, these intermediate results will be several times evicted from the CPU cache in favor of other intermediate results. Consequently, many cache misses during such a computation will occur.

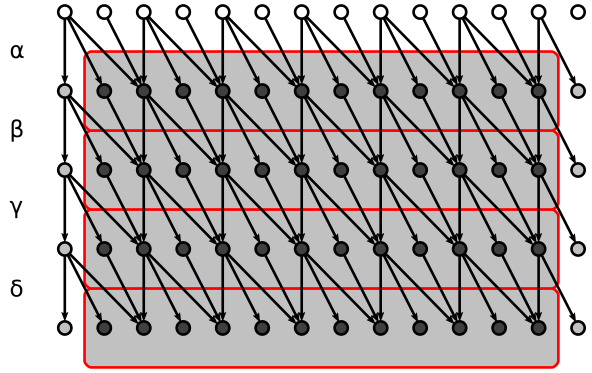

Figure 3: *The horizontal vectorisation of the CDF 9/7 data flow graph. The scaling of coefficients was omitted. The computation within the highlighted areas can be processed in parallel.*

In this paper, this method is called the horizontal vectorisation. This name reflects the fact that the data flow graph is split in horizontal areas as in Figure 3. In each area, the elementary calculations are independent and can be computed in parallel. For simplicity, the scaling of coefficients and the prolog and epilog phases were omitted in the referenced figure. An entire signal of $2S$ samples must be loaded into the memory which is not suitable for memory limited systems.

4.2 Vertical vectorisation

Another way of lifting data flow graph evaluation is the double-loop approach [7]. This approach is referred to as the vertical vectorisation. Earlier, it was described in [4] focusing on low memory systems but without vectorisation.

The P_n and U_n filters need not be causal. In general, non-causal systems requires storing the whole input signal into memory (as can be seen from Figure 3). This is not suitable for fast or memory limited signal processing nor for a vectorisation. Therefore, it would be appropriate to convert non-causal lifting steps (P_n and U_n) to causal systems. The key to force these filtering steps to be causal is the introduction of appropriate delays.

$$\mathcal{P}_n(z) = z^{-l_n} P_n(z) = \sum_{i=0}^{g_n+l_n} p_{i-l_n}^{(n)} z^{-i} \tag{3}$$

$$\mathcal{U}_n(z) = z^{-m_n} U_n(z) = \sum_{i=0}^{f_n+m_n} u_{i-m_n}^{(n)} z^{-i} \tag{4}$$

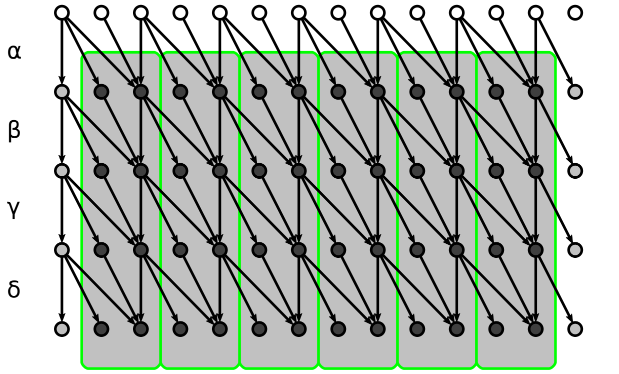

Figure 4: *Vertical vectorisation of the CDF 9/7 data flow graph. The computation within the highlighted areas cannot be processed in parallel due to data dependencies.*

The transition from non-causal to causal system introduce a delay z^{-l_n} on both inputs of the prediction filtering step P_n. In the bottom input s, the delay can be distributed into both branches. This leads to a causal system $\mathcal{P}_n$ as in (3). Analogously, a delay of m_n samples is introduced on both inputs of update step U_n. Again, this delay can distributed into branches of upper input d. The resulting equation is shown in (4). For simplicity, the adjacent delays can combined into single one. Finally in (5), delays of η_n, μ_n and ν_n samples appear around each pair of filtering steps $\mathcal{P}_n$ and $\mathcal{U}_n$. The resulting block diagram is shown in Figure 5.

$$\eta_n = l_n \tag{5a}$$
$$\mu_n = l_n + m_n \tag{5b}$$
$$\nu_n = m_n \tag{5c}$$

In this method, the lifting computation is transformed into one loop instead of multiple loops over all the coefficients. Therefore, one pair of lifting coefficients s_l and d_l is computed in each iteration of such a single loop. However, the computations within each of these areas cannot be directly parallelized due to data dependencies. Even so, this procedure is advantageous because the coefficients are read and written only once. Consequently, this prevents unnecessary cache misses. In our 1-D case, the SIMD vectorisation of this method lies in processing of several adjacent areas in parallel like in [8]. The data flow graph is split in vertical areas of width of two coefficients as in Figure 4. Furthermore, this approach is particularly useful for multidimensional (e.g. 2-D) transform on PC platform where several data rows are processed in single loop at once using n-fold SIMD instructions.

5 Results

The implementations of the approaches described in the previous section was compared on ASVP platform. This comparison was performed on forward DWT using CDF 9/7 wavelet. All the implementations work over a sequence of single-precision floating point numbers. According to platform performance, a length of the sequence was progressively extended from vector of 32 samples with geometrical step of 1.28 up to 240 thousands of samples. The transform was computed including a final coefficient scaling.

Our configuration contains 32-bit MicroBlaze as host CPU and two BCE acceleration units (only one used for 1-D transform). Used bitstream fits into Spartan-6 SP605 FPGA kit. We use PetaLinux as operating system on host CPU. The evaluated programs had been compiled by GCC 3.4.1 with `-O2 -mno-xl-soft-mul -mhard-float` options.

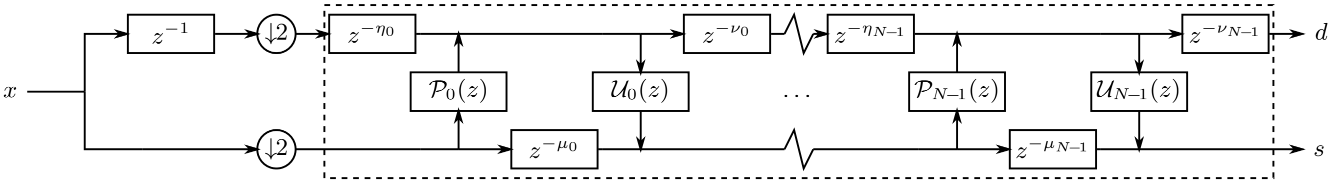

Figure 5: *Block diagram of vertical lifting scheme vectorisation. The part bounded with dashed line correspond to the area of parallel computation.*

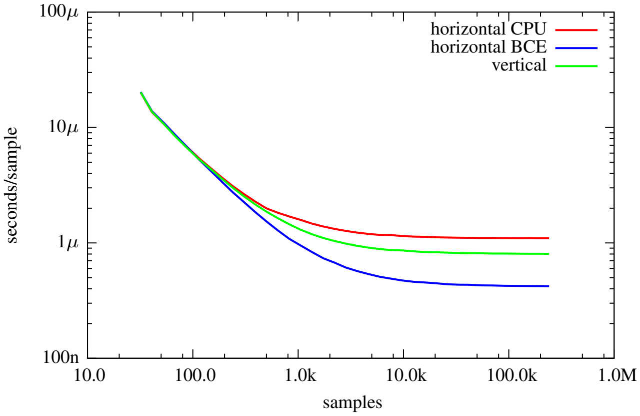

Figure 6: *Comparison of three described approaches on the ASVP platform. The horizontal parallelization was implemented on the CPU as well as on the BCE unit. Using the BCE unit, horizontal parallelization is clearly the fastest method with up to* 2.6× *speedup.*

Evaluation on ASVP platform is summarized in Figure 6. The horizontal axis of this graph indicates the sequence length. The vertical axis specifies computation time per one signal sample. Both approaches from previous section as implemented on MicroBlaze CPU are plotted in this graph. Furthermore, another implementation of the horizontal parallelization accelerated using BCE unit is plotted here. Clearly, the horizontal parallelization is the fastest method when BCE is used. Without BCE, the fastest approach seems to be the vertical parallelization. The speedup of horizontal approach with BCE over baseline horizontal approach on CPU is up to 2.6×.

vectorisation	t samples	b coefficients	q operations
horizontal	$2S$	$2S$	S
vertical	$2T$	$2N$	T

Table 1: *Memory consumption of vectorisation methods. Each method needs t samples to start iteration and b memory words to pass intermediate results between them. In each iteration, up to q operations can be evaluated in parallel.*

vectorisation	μs/sample	speedup
CPU horizontal	1.1	1.0
CPU vertical	0.8	1.4
BCE horizontal	**0.4**	**2.6**

Table 2: *Execution times per sample measured for 240 thousands of samples. All times are related to the CDF 9/7 transform.*

6 Conclusion

In this paper, two known methods of lifting scheme evaluation was compared on ASVP platform. The achieved speedup is up to 2.6× using Application Specific Vector Processor. The best results were obtained using horizontal parallelization performed on one BCE computing unit. This unit can accelerate operations on vectors with up to 1024 elements of length. Operations on longer vectors have to be split into several fragments. The resulting firmware consists of 15 individual operation calls (addition, multiplication) including scaling of lifting coefficients.

Our next research will focus to an adaptation of the proposed approach to the 2-D wavelet transform. Specifically, we will use the single-loop approach proposed by R. Kutil in [7].

Acknowledgements

This work has been supported by the EU FP7-ARTEMIS project IMPART (grant no. 316564) and the national TAČR project RODOS (code TE01020155).

References

[1] R. Bartosinski, M. Daněk, J. Sýkora, L. Kohout, and P. Honzík. Foreground detection and image segmentation in a flexible ASVP platform for FPGAs. In *Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 1–2, 2012.

[2] R. Bartosinski, M. Daněk, J. Sýkora, L. Kohout, and P. Honzík. Video surveillance application based on application specific vector processors. In *Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 1–8, 2012.

[3] C. Chrysafis and A. Ortega. Line-based, reduced memory, wavelet image compression. *IEEE Transactions on Image Processing*, 9(3):378–389, 2000.

[4] C. Chrysafis and A. Ortega. Minimum memory implementations of the lifting scheme. In *Proceedings of SPIE, Wavelet Applications in Signal and Image Processing VIII*, volume 4119 of *SPIE*, pages 313–324, 2000.

[5] A. Cohen, I. Daubechies, and J.-C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 45(5):485–560, 1992.

[6] I. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. *Journal of Fourier Analysis and Applications*, 4(3):247–269, 1998.

[7] R. Kutil. A single-loop approach to SIMD parallelization of 2-D wavelet lifting. In *Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 413–420, 2006.

[8] R. Kutil, P. Eder, and M. Watzl. SIMD parallelization of common wavelet filters. In *Parallel Numerics '05*, pages 141–149, 2005.

[9] S. Mallat. *A Wavelet Tour of Signal Processing: The Sparse Way. With contributions from Gabriel Peyré.* Academic Press, 3 edition, 2009.

[10] J. Sýkora, R. Bartosinski, L. Kohout, M. Daněk, and P. Honzík. Reducing instruction issue overheads in Application-Specific Vector Processors. In *Proceedings of the 15th Euromicro Conference on Digital System Design (DSD)*, DSD '12, pages 600–607, 2012.

[11] J. Sýkora, L. Kohout, R. Bartosinski, L. Kafka, M. Daněk, and P. Honzík. The architecture and the technology characterization of an FPGA-based customizable Application-Specific Vector Processor. In *IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, pages 62–67, 2012.