VECTORISATION OF WAVELET LIFTING

David Barina and Pavel Zemcik
Faculty of Information Technology, Brno University of Technology
{ibarina,zemcik}@fit.vutbr.cz

Introduction

With the start of the widespread use of discrete wavelet transform the need for its effective implementation is becoming increasingly more important. This work presents a novel approach to discrete wavelet transform through a new computational scheme of wavelet lifting. The presented approach is compared with two other. The results are obtained on a general purpose processor with 4-fold SIMD instruction set (such as Intel x86-64 processors). Using the frequently exploited CDF 9/7 wavelet, the achieved speedup is about 3 times compared to naive implementation.

Wavelet transform

The discrete wavelet transform (DWT) is able to decompose discrete signal into lowpass and highpass frequency components. 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 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.

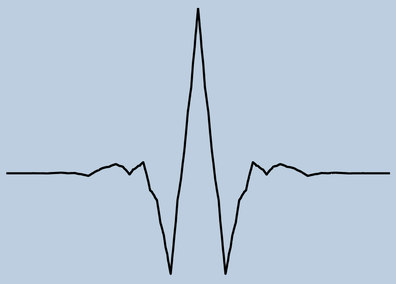


Figure 1: The used CDF 9/7 wavelet.

Lifting scheme

According to the number of arithmetic operations, the lifting scheme 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 $u_i^{(n)}$.

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

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

In this example, the individual lifting steps use 2-tap symmetric filters for the prediction as well as the update.

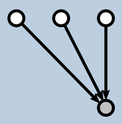


Figure 2: The elementary operation.

Acknowledgements

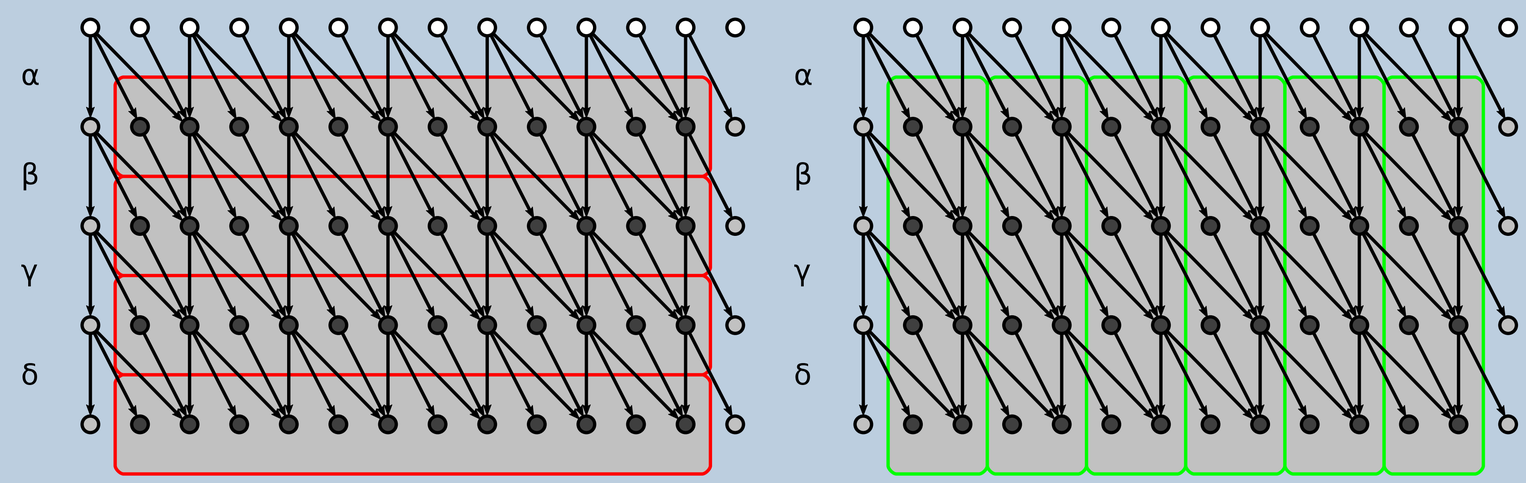
This work has been supported by the EU FP7-ARTEMIS project IMPART (grant no. 316564) and the national Technology Agency of the Czech Republic project RODOS (no. TE01020155).

Lifting scheme vectorisation

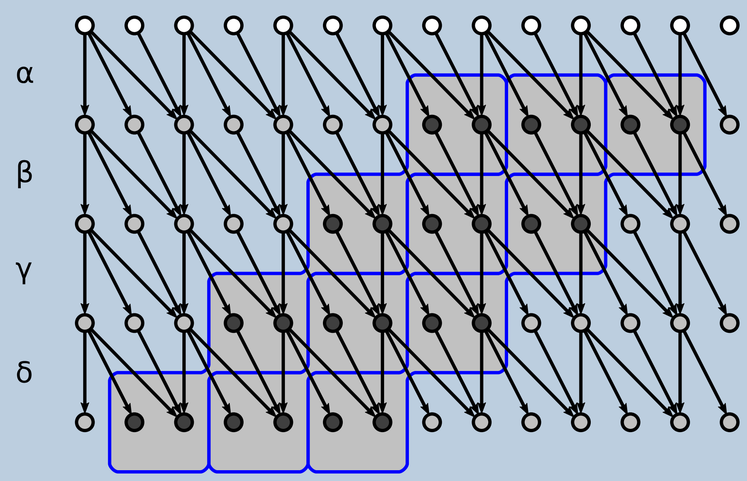
The calculation scheme described in the previous section can be realized in a number of different ways. The main difference between them is in the order of lifting steps evaluation.

Horizontal vectorisation The naive approach of data flow graph evaluation directly follows the lifting steps. Thus, all intermediate coefficients in first row are evaluated in the first step, etc.

Vertical vectorisation 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 is computed in each iteration of such a single loop.



Proposed method Unlike the vertical approach, the elementary lifting operations evaluated in single loop iterations are shifted with respect to each other. This shift removes the data dependency within these loop iteration. Therefore, the elementary operations can be now computed in parallel.



Evaluation

The implementations of the approaches described in the previous section was evaluated on two x86-64 platforms. Both processors are equipped with 4-fold SSE instruction set. The first platform used in this paper is a Intel Core2 Duo CPU E7600 at 3.06 GHz with 32 kB of L1 8-way set associative data cache and 3 MB of L2 cache. Measurement results were verified on an AMD Athlon 64 X2 4000+ at 2.1 GHz with 64 kB L1 2-way set associative data cache and 512 kB of L2 cache.

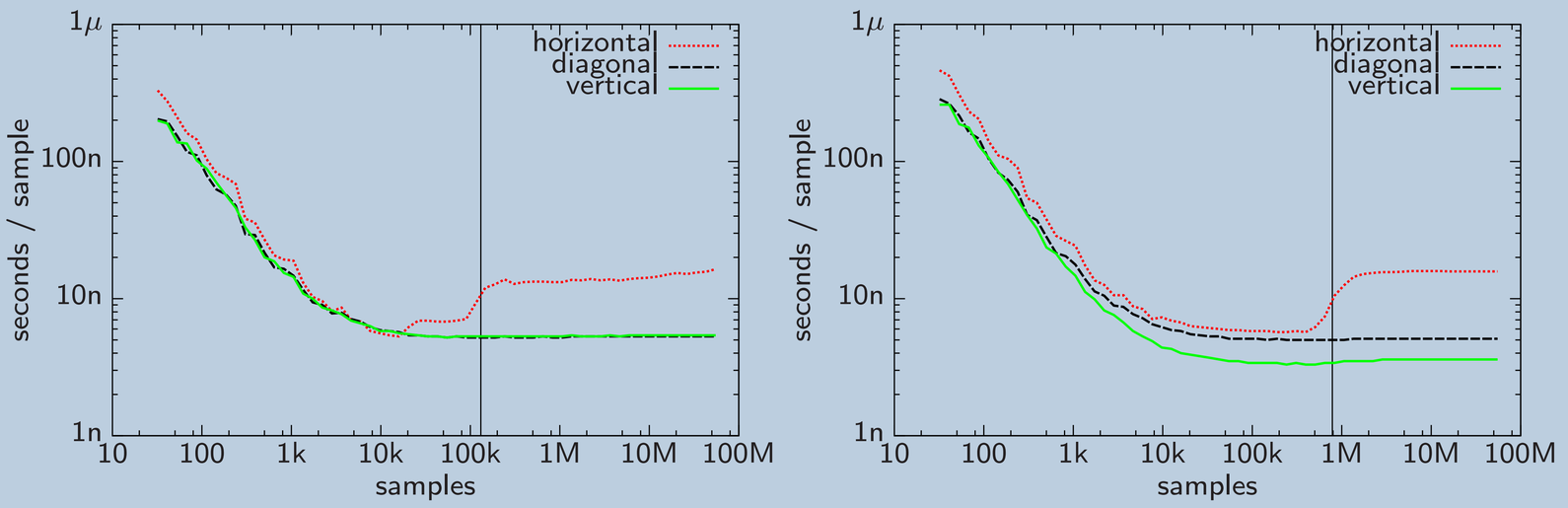


Figure 3: The AMD Athlon 64 X2 (left) and Intel Core2 Duo (right).

Summary

Table shows the comparison of the different algorithms in terms of memory consumption. Each of the method require t samples to start iteration of the vectorised loop and b memory cells to store intermediate results. In each iteration, up to q operations can be evaluated in parallel.

vectorisation	t samples	b coefficients	q operations
horizontal	$2S$	$2S$	S
vertical	$2T$ (8)	$2N$ (4)	T (4)
diagonal	2 (2)	$6N-2$ (10)	$2N$ (4)