Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE @ DIRECT®

Signal Processing 84 (2004) 1689–1699

SIGNAL PROCESSING

www.elsevier.com/locate/sigpro

Efficient cache-based spatial combinative lifting algorithm for wavelet transform

Chun-Kuang Hu[a], Wen-Ming Yan[a], Kuo-Liang Chung[b,*,1]

[a] *Department of Computer Science and Information Engineering, National Taiwan University, No. 1, Section 4, Roosevelt Road, Taipei 10764, Taiwan*

[b] *Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, No. 43, Section 4, Keelung Road, Taipei 10672, Taiwan*

Received 31 May 2002; received in revised form 2 April 2004

Abstract

Recently, a fast spatial combinative lifting algorithm (SCLA) for performing the wavelet transform (WT) was presented by Meng and Wang, and the SCLA speeds up the well-known lifting scheme presented by Daubechies and Sweldens. Employing the concept in the block-based WT by Bao and Kuo, this letter presents an efficient cache-based SCLA (CSCLA) for performing the WT. Theoretically, the number of total arithmetical operations required in the proposed CSCLA is equal to that in the SCLA. Experimental results confirm the computational advantage of our proposed CSCLA when compared to some other previous results. In addition, the VTune Performance Analyzer is used to evaluate the cache performance among the concerning algorithms.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Cache-based algorithm; Spatial combinative lifting algorithm; Wavelet transform

1. Introduction

The biorthogonal wavelet $\frac{9}{7}$ filter proposed by Cohen et al. [6] is used in the JPEG2000 standard [4]. Previously, Mallat [11] presented a convolution-based implementation for performing the WT. The convolution-based implementation takes $9n^2$ multiplications and $14n^2$ additions where the image size is of $n \times n$. Later Daubechies and Sweldens [7] presented a lifting scheme-based implementation which takes $6n^2$ multiplications and $8n^2$ additions and is faster than the convolution-based implementation. Recently, Meng and Wang [12] presented an efficient spatial combinative lifting algorithm (SCLA) for performing the WT in order to improve the lifting scheme-based WT. The SCLA only takes $3.5n^2$ multiplications and $8n^2$ additions. From the comparison of the number of operations required, the SCLA is the fastest among the three methods. Under the cache memory environment [8], some cache—and VLSI-based WTs [3,10] and cache—and convolution-based WTs [2,5] have been developed in the literatures. Among the cache- and convolution-based WTs, Bao and Kuo

* Corresponding author. Tel.: +866-2-27376771; fax: +886-2-27376777.
E-mail address: klchung@cs.ntust.edu.tw (K. Chung).

[1] Supported by the National Science Council of R.O.C. under contract NSC90-2213-E011-056.

0165-1684/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.sigpro.2004.05.014

[2] presented the block-based WT to improve the line-based WT [5]. Especially, once the WT of one block has been finished, then a portion of the resulting block can be sent to the image codec immediately. Since the SCLA is faster than the convolution-based implementation for performing the WT, the motivation of this letter is to present an efficient cache-based SCLA (CSCLA) by employing the block-based approach [2].

According to the special computational structure of the SCLA, we first analyze the data dependence in the SCLA, then we present an efficient CSCLA. Theoretically, the number of total arithmetical operations is equal to that in the SCLA. Experimental results confirm the computational advantage of our proposed CSCLA when compared to some other previous results [2,5,12]. In addition, the VTune Performance Analyzer is used to evaluate the cache performance among the concerning algorithms. The cache performance evaluation also reveals the memory-access advantage of our proposed CSCLA.

2. The SCLA

In this section, we first introduce the lifting scheme [7], and then we introduce the SCLA [12]. Let the input image of size $n \times n$ be denoted by the matrix X where

$$X = \begin{pmatrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdot & \cdot & \cdot & x_{0,n-1} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdot & \cdot & \cdot & x_{1,n-1} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdot & \cdot & \cdot & x_{2,n-1} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ x_{n-1,0} & x_{n-1,1} & x_{n-1,2} & \cdot & \cdot & \cdot & x_{n-1,n-1} \end{pmatrix}$$

and let the $\frac{9}{7}$ filter T be denoted by

$$T = \begin{pmatrix} h_0 & 2h_1 & 2h_2 & 2h_3 & 2h_4 & & & & & & \\ g_1 & g_0+g_2 & g_1+g_3 & g_2 & g_3 & & & & & & \\ h_2 & h_1+h_3 & h_0+h_4 & h_1 & h_2 & h_3 & h_4 & & & & \\ g_3 & g_2 & g_1 & g_0 & g_1 & g_2 & g_3 & & & & \\ h_4 & h_3 & h_2 & h_1 & h_0 & h_1 & h_2 & h_3 & h_4 & & \\ & & & & & \cdot & & & & & \\ & & & & & & \cdot & & & & \\ & & & & & & & \cdot & & & \\ & & & & & h_4 & h_3 & h_2+h_4 & h_1+h_3 & h_0+h_2 & h_1 \\ & & & & & & & 2g_3 & 2g_2 & 2g_1 & g_0 \end{pmatrix},$$

where $h_0 = 0.60295$, $h_1 = -0.26686$, $h_2 = -0.07822$, $h_3 = 0.01686$, $h_4 = 0.02675$, g_0=1.11509, $g_1 = 0.59127$, $g_2 = -0.05754$, and $g_3 = -0.09127$. Using the above filter T, the WT of X can be calculated by

$$Y = TXT^t. \tag{1}$$

From the result in [7], T can be factorized into $T = EDCBA$ where

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \alpha & 1 & \alpha & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & \alpha & 1 & \alpha & \cdots & 0 & 0 & 0 \\ & & & & \cdots & & & & \\ & & & & & \cdots & & & \\ 0 & 0 & 0 & 0 & \cdots & \alpha & 1 & \alpha & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 2\alpha & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 2\beta & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \beta & 1 & \beta & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ & & & & \cdots & & & & \\ & & & & & \cdots & & & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & \beta & 1 & \beta \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{pmatrix},$$

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \gamma & 1 & \gamma & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & \gamma & 1 & \gamma & \cdots & 0 & 0 & 0 \\ & & & & \cdots & & & & \\ & & & & & \cdots & & & \\ 0 & 0 & 0 & 0 & \cdots & \gamma & 1 & \gamma & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 2\gamma & 1 \end{pmatrix}, \quad D = \begin{pmatrix} 1 & 2\delta & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \delta & 1 & \delta & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ & & & & \cdots & & & & \\ & & & & & \cdots & & & \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & \delta & 1 & \delta \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{pmatrix},$$

$$\text{and } E = \begin{pmatrix} \xi & & & & & & \\ & -\xi^{-1} & & & & & \\ & & \xi & & & & \\ & & & -\xi^{-1} & & & \\ & & & & \cdots & & \\ & & & & & \xi & \\ & & & & & & -\xi^{-1} \end{pmatrix},$$

where $\alpha = -1.586134342$, $\beta = -0.0529801185$, $\gamma = 0.8829110762$, $\delta = 0.4435068522$, and $\xi = 1.149604398$. Instead of performing TXT^t directly, the lifting scheme performs $((((((E(D(C(B(AX)))))A^t)B^t)C^t)D^t)E^t)$ and it leads to some computational advantage.

By $T = EDCBA$ and Eq. (1), the SCLA performs the WT of X as follows:

$$\begin{aligned} Y &= TXT^t \\ &= E(D(C(B(AXA^t)B^t)C^t)D^t)E^t. \end{aligned} \tag{2}$$

In order to characterize the computational structure of the SCLA, let $Y^A = AXA^t$, $Y^B = BY^AB^t$, $Y^C = CY^BC^t$, $Y^D = DY^CD^t$, and $Y^E = EY^DE^t$, where $Y^E = Y$.

From $Y^A = AXA^t$, we have

$$y^A_{2i,2j} = x_{2i,2j}, \qquad y^A_{2i,2j+1} = x_{2i,2j+1} + \alpha(y^A_{2i,2j} + y^A_{2i,2j+2}),$$

$$y^A_{2i+1,2j+1} = x_{2i+1,2j+1} + \alpha(x_{2i+1,2j} + x_{2i+1,2j+2} + y^A_{2i,2j+1} + y^A_{2i+2,2j+1}),$$

$$y^A_{2i+1,2j} = x_{2i+1,2j} + \alpha(y^A_{2i,2j} + y^A_{2i+2,2j}). \tag{3}$$

Assume $y^A_{i,-1} = y^A_{i,1}$ and $y^A_{-1,i} = y^A_{1,i}$ for $i = -1, 0, 1, \ldots, n-1$. In order to use the temporary results of Y^A in computing Y^B, we change the calculation "square" from $[(2i,2j),(2i,2j+1),(2i+1,2j),(2i+1,2j+1)]$ to $[(2i-1,2j-1),(2i-1,2j),(2i,2j-1),(2i,2j)]$. For computing $Y^B = BY^AB^t$, we can derive that $y^B_{2i-1,2j-1} = y^A_{2i-1,2j-1}$, $y^B_{2i-1,2j} = y^A_{2i-1,2j} + \beta(y^B_{2i-1,2j-1} + y^B_{2i-1,2j+1})$, $y^B_{2i,2j} = y^A_{2i,2j} + \beta(y^A_{2i,2j-1} + y^A_{2i,2j+1} + y^B_{2i-1,2j} + y^B_{2i+1,2j})$, and $y^B_{2i,2j-1} = y^A_{2i,2j-1} + \beta(y^B_{2i-1,2j-1} + y^B_{2i+1,2j-1})$. Because the structures of matrices C and D are similar to those of matrices A and B, the calculation of the multiplications with matrices C and D is following exactly the steps used for A and B with a cascade replacement of the input by the output of the previous step.

Since the computational structure of Y^A, Y^B, Y^C, and Y^D are similar, we only list the procedure for computing $Y^A = AXA^t$ for saving the space of the context. Initially we perform $z_{i,j} = x_{i,j}$. The final result $y^A_{i,j}$ is saved into $z_{i,j}$, i.e. $z_{i,j} = y^A_{i,j}$.

```
for $i \leftarrow 0$ to $n-1$
    $z_{i,n} \leftarrow z_{i,n-2}$
for $j \leftarrow 0$ to $n$
    $z_{n,j} \leftarrow z_{n-2,j}$
for $i \leftarrow 0$ to $\frac{n}{2}$
    for $j \leftarrow 0$ to $\frac{n}{2}-1$
        $z_{2i,2j+1} \leftarrow z_{2i,2j+1} + \alpha(z_{2i,2j} + z_{2i,2j+2})$
for $i \leftarrow 0$ to $\frac{n}{2}-1$
    for $j \leftarrow 0$ to $\frac{n}{2}-1$
$z_{2i+1,2j+1} \leftarrow z_{2i+1,2j+1} + \alpha(z_{2i+1,2j} + z_{2i+1,2j+2} + z_{2i,2j+1} + z_{2i+2,2j+1})$
for $i \leftarrow 0$ to $\frac{n}{2}-1$
    for $j \leftarrow 0$ to $\frac{n}{2}$
        $z_{2i+1,2j} \leftarrow z_{2i+1,2j} + \alpha(z_{2i,2j} + z_{2i+2,2j})$
```

In the above procedure, it is easy to verify that $\frac{3}{4}n^2$ multiplications and $2n^2$ additions are needed for computing Y^A.

Finally we want to compute $Y = EY^DE^t$. Trivially we have $y_{2i,2j} = \xi s_{2i,2j} = \xi^2 y^D_{2i,2j}$, $y_{2i,2j+1} = \xi s_{2i,2j+1} = -y^D_{2i,2j+1}$, $y_{2i+1,2j} = -\xi^{-1}s_{2i+1,2j} = -y^D_{2i+1,2j}$, and $y_{2i+1,2j+1} = -\xi^{-1}\ s_{2i+1,2j+1} = \xi^{-2}y^D_{2i+1,2j+1}$. The following procedure is used to perform $Y^E = EY^DE^t$ and it takes $\frac{1}{2}n^2$ multiplications.

```
for $i \leftarrow 0$ to $\frac{n}{2} - 1$
  for $j \leftarrow 0$ to $\frac{n}{2} - 1$
    $z_{2i,2j} \leftarrow \xi^2 z_{2i,2j}$; $z_{2i+1,2j} \leftarrow -z_{2i,2j}$; $z_{2i,2j+1} \leftarrow -z_{2i,2j}$; $z_{2i+1,2j+1} \leftarrow \xi^{-2} z_{2i,2j}$
```

Summing up the total number of multiplications and additions required in the the computations, Y^A, Y^B, Y^C, Y^D, and Y^E, it takes $3.5n^2 (= \frac{3}{4}n^2 + \frac{3}{4}n^2 + \frac{3}{4}n^2 + \frac{3}{4}n^2 + \frac{1}{2}n^2)$ multiplications and $8n^2$ additions.

3. The proposed cache-based SCLA: CSCLA

In this section, we describe our proposed CSCLA. In order to increase the performance of the SCLA in the cache memory, following the block-based approach [2], we first partition the input image X into some smaller blocks and then we perform the SCLA on one block each time in the cache environment. During performing the SCLA on one block, we need to consider the overlapped areas between that block and its north and west neighbors. In Fig. 1, we assume that the size of each block is $M \times M$ and a sliding window is used to bound a region covering the needed pixels to perform the SCLA on that block. In order to perform the CSCLA successfully, we must keep additional 5 columns (rows) in the west (south) neighboring block. In Fig. 1, when the sliding window has been shifted from block a to block b, we thus have a $(M + 5) \times 5$ area that is overlapped with the previous sliding window. When shifting the sliding window to block d, we have a $5 \times (M + 5)$ horizontal overlapped area. By the same way, when shifting the sliding window to block e, we have an $M \times 5$ vertical overlapped area and a $5 \times M$ horizontal overlapped area. After the horizontal overlapped area and the vertical overlapped area have been taken into account, we thus can perform the SCLA on any one block successfully.

Fig. 2, which is quite similar to that in [2], illustrates how the proposed CSCLA is performed. The CSCLA is composed of nine cases to be considered. Among these nine cases, case 1 case 3 case 7 and case 9 are concerning with four corner cases; case 2 case 4 case 6 and case 8 are concerning with four boundary cases, and all the remaining blocks belong to case 5. According to the raster scan order, the horizontal overlapped area of each block must be kept in an additional buffer which will be used by the south neighboring block in the next block row. All the vertical overlapped area is not required to be saved in the buffer since the overlapped area can be used in the next block immediately. For each case of Fig. 2, the light shaded area is unnecessary to be saved in the buffer while the dark shaded area must be kept in the additional memory. For exposition, we define two types of memory which are used in our proposed CSCLA. The first type of memory is the working memory which is used to perform the SCLA in a block and the size of working memory equals to the size of sliding window, i.e. $(M + 5) \times (M + 5)$. The second type of memory is called the row buffer and we use this row buffer to keep the horizontal overlapped area of a whole block row. The size of the row buffer is $5 \times N$ where N denotes the width of the input image.

In order to describe the computational flow of the proposed CSCLA, we demonstrate some simulation snapshots to demonstrate how the CSCLA works. For simplicity, we only simulate one stage WT although it can be extended to the multistage WT. In our experiments (see Section 3), we compare the performance among our proposed CSCLA and the related previous methods by carrying out four-stage WT and inverse WT. Here only case 5 (see Fig. 2) of the CSCLA is investigated since the remaining eight cases are similar to case 5 but involve the mirroring process on the boundary pixels. First, the initial working memory is

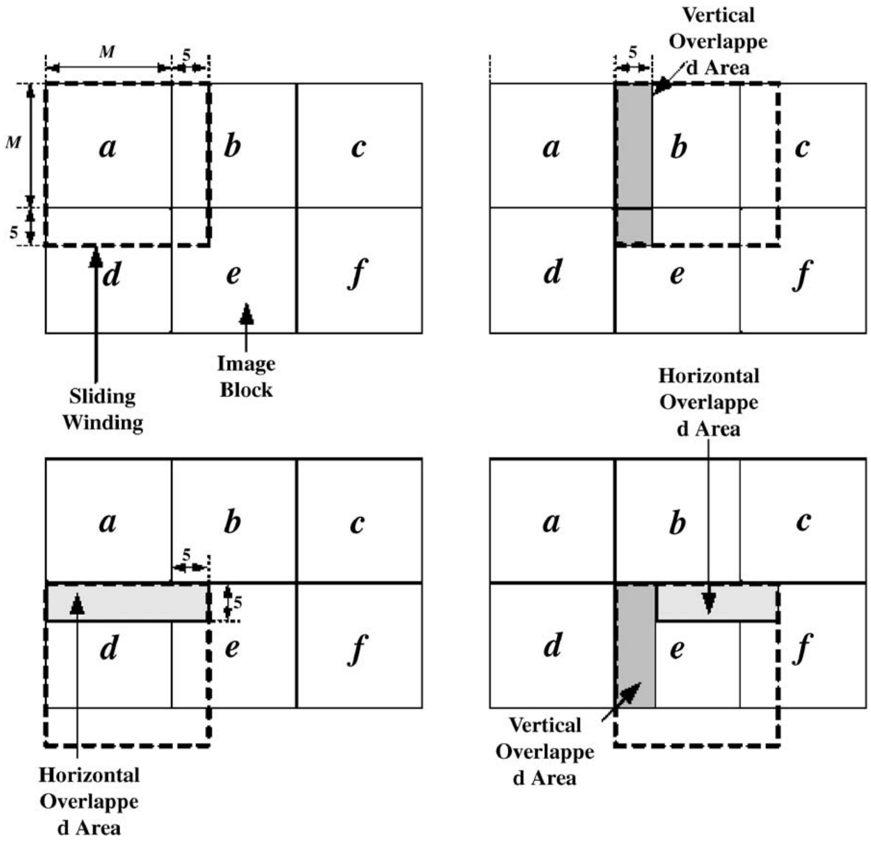

Fig. 1. The concerning overlapped areas.

given by

$$W = \begin{pmatrix} D & D & D & D & D & . & D \\ D & D & D & D & D & . & D \\ D & C & C & C & C & . & C \\ D & C & B & B & B & . & B \\ D & C & B & A & A & . & A \\ D & C & B & A & X & . & X \\ . & . & . & . & . & . & . \\ D & C & B & A & X & . & X \\ C & C & B & A & X & . & X \\ B & B & B & A & X & . & X \\ A & A & A & A & X & . & X \end{pmatrix},$$

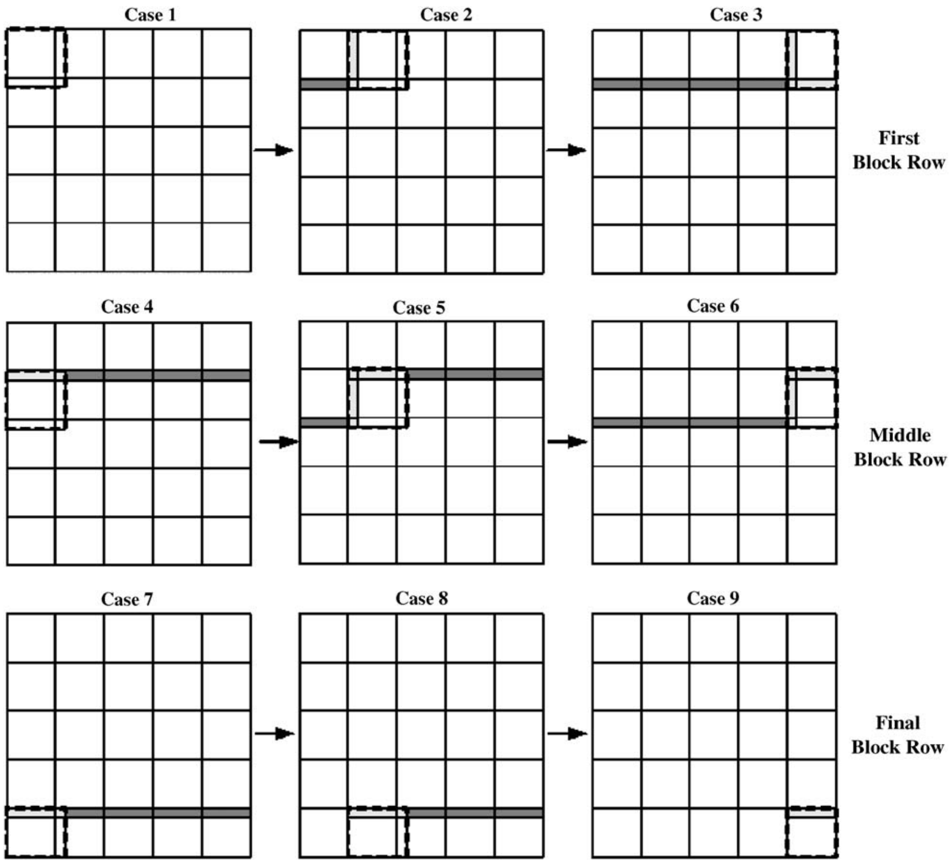

Fig. 2. The nine cases of the CSCLA.

where W denotes a $(M+5)\times(M+5)$ working memory; X denotes one pixel of the block with size $M\times M$, and A, B, C, and D denote the previously calculated wavelet coefficients, i.e. Y^A, Y^B, Y^C, and Y^D, respectively. $W[0..M+4,0..4]$ and $W[0..4,5..M+4]$ denote the vertical and horizontal overlapped areas, respectively.

From the data dependence (see Eq. (3)) in the computation $Y^A = AXA^t$, we begin with performing $Y^A[5..M+4,4..M+3]$ in the cache memory and the resulting snapshot of W is shown below:

$$\begin{pmatrix} D & D & D & D & D & . & D \\ D & D & D & D & D & . & D \\ D & C & C & C & C & . & C \\ D & C & B & B & B & . & B \\ D & C & B & A & A & . & A \\ D & C & B & A & X & . & X \\ . & . & . & . & . & . & . \\ D & C & B & A & X & . & X \\ C & C & B & A & X & . & X \\ B & B & B & A & X & . & X \\ A & A & A & A & X & . & X \end{pmatrix} \to \begin{pmatrix} D & D & D & D & . & D & D \\ D & D & D & D & . & D & D \\ D & C & C & C & . & C & C \\ D & C & B & B & . & B & B \\ D & C & B & A & . & A & A \\ D & C & B & A & . & A & X \\ . & . & . & . & . & . & . \\ D & C & B & A & . & A & X \\ C & C & B & A & . & A & X \\ B & B & B & A & . & A & X \\ A & A & A & A & . & A & X \end{pmatrix}.$$

Then we perform $Y^B = BMB^t$, $Y^C = CMC^t$, $Y^D = DMD^t$, and $Y^E = EME^t$ based on the data dependence mentioned in the SCLA (see Section 2); the related four snapshots of W's are shown below:

$$\begin{pmatrix} D & D & D & D & . & D & D \\ D & D & D & D & . & D & D \\ D & C & C & C & . & C & C \\ D & C & B & B & . & B & B \\ D & C & B & A & . & A & A \\ D & C & B & A & . & A & X \\ . & . & . & . & . & . & . \\ D & C & B & A & . & A & X \\ C & C & B & A & . & A & X \\ B & B & B & A & . & A & X \\ A & A & A & A & . & A & X \end{pmatrix} \to \begin{pmatrix} D & D & D & . & D & D & D \\ D & D & D & . & D & D & D \\ D & C & C & . & C & C & C \\ D & C & B & . & B & B & B \\ D & C & B & . & B & A & A \\ D & C & B & . & B & A & X \\ . & . & . & . & . & . & . \\ D & C & B & . & B & A & X \\ C & C & B & . & B & A & X \\ B & B & B & . & B & A & X \\ A & A & A & . & A & A & X \end{pmatrix} \to \begin{pmatrix} D & D & . & D & D & D & D \\ D & D & . & D & D & D & D \\ D & C & . & C & C & C & C \\ D & C & . & C & B & B & B \\ D & C & . & C & B & A & A \\ D & C & . & C & B & A & X \\ . & . & . & . & . & . & . \\ D & C & . & C & B & A & X \\ C & C & . & C & B & A & X \\ B & B & . & B & B & A & X \\ A & A & . & A & A & A & X \end{pmatrix}$$

$$\to \begin{pmatrix} D & . & D & D & D & D & D \\ D & . & D & D & D & D & D \\ D & . & D & C & C & C & C \\ D & . & D & C & B & B & B \\ D & . & D & C & B & A & A \\ D & . & D & C & B & A & X \\ . & . & . & . & . & . & . \\ D & . & D & C & B & A & X \\ C & . & C & C & B & A & X \\ B & . & B & B & B & A & X \\ A & . & A & A & A & A & X \end{pmatrix} \to \begin{pmatrix} Y & . & Y & D & D & D & D & D \\ Y & . & Y & D & D & D & D & D \\ Y & . & Y & D & C & C & C & C \\ Y & . & Y & D & C & B & B & B \\ Y & . & Y & D & C & B & A & A \\ Y & . & Y & D & C & B & A & X \\ . & . & . & . & . & . & . & . \\ Y & . & Y & D & C & B & A & X \\ D & . & D & D & C & B & A & X \\ D & . & D & D & C & B & A & X \\ C & . & C & C & C & B & A & X \\ B & . & B & B & B & B & A & X \\ A & . & A & A & A & A & A & X \end{pmatrix},$$

where Y's denote the one stage wavelet coefficients. It is observed that the above cache-based computation in the CSCLA fully utilize the locality advantage of the current block in the cache memory. Up to here, the partial final wavelet coefficients, i.e. $W[0..M, 0..M]$, has been obtained, and it can be pumped out. $W[M..M+4, 0..M]$ must be moved to the row buffer and they will be used as the horizontal overlapped area in the south neighboring block. For performing the CSCLA on the east neighboring block, we shift $W[0..M+4, M..M+4]$ to $W[0..M+4, 0..4]$, then feed $M \times M$ X's in the next block into $W[5..M+4, 5..M+4]$ and move temporary the last five rows of the north neighboring block into the vertical overlapped area, i.e. $W[0..4, 5..M+4]$, we thus obtain

$$W = \begin{pmatrix} D & D & D & D & D & . & D \\ D & D & D & D & D & . & D \\ D & C & C & C & C & . & C \\ D & C & B & B & B & . & B \\ D & C & B & A & A & . & A \\ D & C & B & A & X & . & X \\ . & . & . & . & . & . & . \\ D & C & B & A & X & . & X \\ C & C & B & A & X & . & X \\ B & B & B & A & X & . & X \\ A & A & A & A & X & . & X \end{pmatrix}$$

once again. Repeating the above process continually, the proposed CSCLA can complete one stage WT on whole input image. In fact, the next stage of WT can be performed by the same argument.

From the above description, theoretically, the number of total arithmetical operations required in our proposed CSCLA is equal to that in the SCLA. However, besides inheriting the computational advantage of SCLA, our proposed CSCLA utilizes the fast accessing capability of the cache memory environment.

4. Experimental results

In this section, some experimental results are demonstrated to justify the computational advantage of our proposed CSCLA. All the experiments are performed on the Pentium III computer with 800 MHz and the size of the main memory, L1 cache memory, and L2 cache memory are 128 MB, 32 KB, and 256 KB, respectively. In our experiments, we compare the performance among our proposed CSCLA and the related previous methods by carrying out four-stage WT and inverse WT. The experimental results are shown in Table 1. The first row of Table 1 denotes the size of the input image. The second row and the third row show that the execution-time of two previous cache-based WT in terms of seconds, and they are the line- and convolution-based WT [5] and the block- and convolution-based WT [2], respectively. The execution-time of SCLA is listed in fourth row. Finally, the time performance of our proposed CSCLA is shown in the fifth row. The experimental results reveal that our proposed CSCLA has 50–68% execution-time improvement ratio when compared to the SCLA. Moreover, when compared to the previous two cache-based methods, our proposed CSCLA still has 50% execution-time improvement ratio.

Table 1
Execution-time comparison among our proposed CSCLA and three related methods

Size of the image	256×256	512×512	1024×1024
Convolution + line-based WT	0.016	0.068	0.301
Convolution + block-based WT	0.016	0.068	0.296
SCLA	0.017	0.107	0.434
CSCLA	0.008	0.034	0.17

Table 2
Cache performance among our proposed CSCLA and three related methods

Memory event	L1 CRs	L2 CRs	L2 CRMs
Convolution + line-based WT	2,467,192	113,962	15,019
Convolution + block-based WT	2,251,415	49,921	23,087
SCLA	1,454,253	77,690	73,889
CSCLA	609,787	44,257	20,309

To directly measure cache hits and misses for each concerning algorithm, the VTune Performance Analyzer is used to evaluate the cache performance. In this analyzer, there are three types of memory events, namely the L1 cache requests (L1 CRs), L2 cache requests (L2 CRs), and L2 cache request misses (L2 CRMs), to be used to measure the cache performance. Here one L2 CR is needed when the response of one L1 CR is a miss. For simplicity, we use a 512×512 image as the testing image. Table 2 illustrates the cache performance among the proposed CSCLA and the three concerning algorithms. We first compare the cache performance between the SCLA and the CSCLA. Table 2 reveals that the number of L1 CRs required in the CSCLA is much less than that in the SCLA; the number of L2 CRs required in the CSCLA is still less than that in the SCLA, and the number of L2 CRMs required in the CSCLA is still less than that in the SCLA. It is clear that the proposed CSCLA has better cache performance when compared to the SCLA and it is consistent with the execution-time comparison in Table 1.

Finally, we compare the cache performance among the proposed CSCLA, the block- and convolution-based WT, and the line- and convolution-based WT. From Table 2, it is observed that the number of L1 CRs required in the CSCLA is much less than that in the block- and convolution-based WT; the number of L2 CRs required in the CSCLA is still much less than that in the block- and convolution-based WT, and the number of L2 CRMs required in the CSCLA is also less than that in the block- and convolution-based WT. The better cache performance of our proposed CSCLA also meets the better execution-time performance when compared to the block- and convolution-based WT. Table 2 also reveals the computational advantage of our proposed CSCLA since the number of L1 CRs required in the CSCLA is much less than that in the line- and convolution-based WT; the number of L2 CRs required in the CSCLA is still less than that in the block- and convolution-based WT although the number of L2 CRMs required in the CSCLA is more than that in the line- and convolution-based WT.

5. Conclusions

Following the block-based approach, we have presented the proposed CSCLA to improve the time performance of the SCLA. The experimental results show that our proposed CSCLA is faster than the previous three related methods. Experimental results also confirm the computational advantage of our proposed CSCLA. In addition, the VTune Performance Analyzer has been used to evaluate the cache performance among the concerning algorithms. The cache performance evaluation also reveals the memory-access advantage of our proposed CSCLA. Recently, some new VLSI- and lifting-based WTs have been proposed in [1,9]. It seems that the results of this letter can be applied to the VLSI environment.

References

[1] K. Andra, C. Chakrabarti, T. Acharya, A VLSI architecture for lifting-based forward and inverse wavelet transform, IEEE Trans. Signal Process. 50 (4) (April 2002) 966–977.

[2] Y.L. Bao, C.-C. Jay Kuo, Design of wavelet-based image codec in memory-constrained environment, IEEE Trans. Circuits Syst. Video Technol. 11 (5) (May 2001) 642–650.

[3] C. Chakrabarti, C. Mumford, Efficient realizations of encoders and decoders based on the 2-D discrete wavelet transform, IEEE Trans. VLSI Syst. 7 (3) (September 1999) 289–298.

[4] C. Christopoulos, JPEG2000 Verification Model 4.0 (Technique Description), ISO/IEC JTC1/SC29/WG1 N1282.

[5] C. Chrysafis, A. Ortega, Line-based, reduced memory, wavelet image compression, IEEE Trans. Image Process. 9 (3) (March 2000) 378–389.

[6] A. Cohen, I. Daubechies, J. Feauveau, Biorthogonal Bases of Compactly Supported Wavelets, Commun. Pure Appl. Math. 45 (1992) 485–560.

[7] I. Daubechies, W. Sweldens, Factoring wavelet transforms into lifting steps, J. Fourier Anal. Appl. 4 (3) (1998) 247–269.

[8] J. Hennessy, D. Patterson, Computer Architecture: A Quantitative Approach, Memory-Hierachy Design, Morgan Kaufmann, New York, 1990 (Chapter 8).

[9] W. Jiang, A. Ortega, Lifting factorization-based discrete wavelet transform architecture design, IEEE Trans. on Circuits and Systems for Video Technology 11 (5) (May 2001) 651–657.

[10] G. Lafruit, L. Nachtergaele, J. Bormans, M. Engels, I. Bolsens, Optimal memory organization for scalable texture codecs in MPEG-4, IEEE Trans. Circuits and Syst. Video 9 (2) (March 1999) 218–243.

[11] S. Mallat, A theory for multiresolution signal decomposition: the wavelet representation, IEEE Trans. Pattern Anal. Mach. Intell. 11 (July 1989) 674–693.

[12] H.Y. Meng, Z.H. Wang, Fast spatial combinative lifting algorithm of wavelet transform using the 9/7 filter for image block compression, Electron. Lett. 36 (21) (October 2000) 1766–1767.