Fast Algorithm for Walsh Hadamard Transform on Sliding Windows

Wanli Ouyang, W.K. Cham

***Abstract*— This paper proposes a fast algorithm for Walsh Hadamard Transform on sliding windows which can be used to implement pattern matching most efficiently. The computational requirement of the proposed algorithm is about 4/3 additions per projection vector per sample which is the lowest among existing fast algorithms for Walsh Hadamard Transform on sliding windows.**

***Index Terms*— Fast algorithm, Computation of transforms, Walsh Hadamard Transform, Pattern Matching**

I. Introduction

Pattern matching, also named as template matching, is widely used in signal processing, computer vision, image and video processing. Pattern matching has found application in manufacturing for quality control [1], image based rendering [2], image compression [3], object detection [4] and video compression. The block matching algorithm used for video compression can be considered as a pattern matching problem [5]-[8]. To relieve the burden of high complexity and high requirement of computational time for pattern matching, a lot of fast algorithms have been proposed [9]-[13].

It has been found that pattern matching can be performed efficiently in Walsh Hadamard Transform (WHT) domain [9]. In pattern matching, signal vectors obtained by a sliding window need to be compared to a sought pattern. Hel-Or and Hel-Or's algorithm [9] requires $2N$-2 additions for obtaining all WHT projection values in each window of size N. Note that one subtraction is considered to be one addition regarding the computational complexity in this paper. Their algorithm achieves efficiency by utilizing previously computed values in the internal vertices of the tree structure in Fig. 1. Recently, the Gray Code Kernel (GCK) algorithm [10] which utilizes previously computed values in the leaves of the tree structure in Fig. 1 was proposed. The GCK algorithm requires similar computation as [9] when all projection values are computed and requires less computation when only a small number of projection vectors are computed.

The authors are with the Department of Electronic Engineering, The Chinese University of Hong Kong, (e-mail: wlouyang@ee.cuhk.edu.hk; wkcham@ee.cuhk.edu.hk)

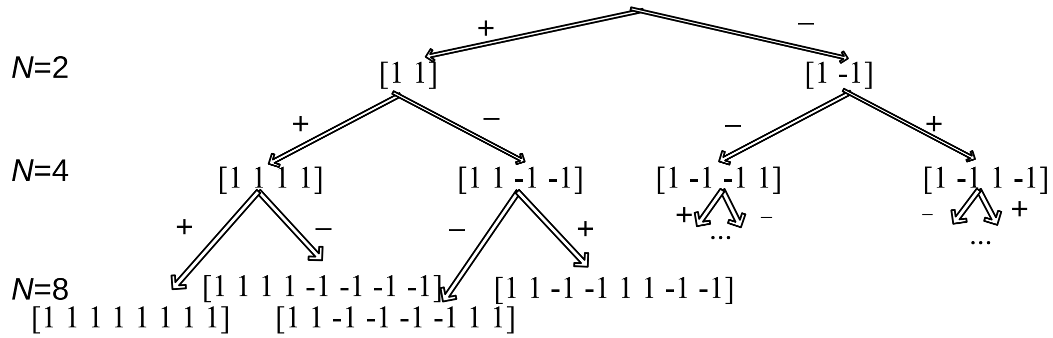


Fig. 1 Tree structure for Walsh-Hadamard Transform in sequency order

This paper proposes a fast algorithm for WHT on sliding windows. Instead of performing order-N WHT by means of order-$N/2$ WHT and N additions in the tree structure, which is the technique adopted in [9], the proposed algorithm computes order-N WHT by means of order-$N/4$ WHT and $N+1$ additions. In this way, the proposed algorithm can obtain all WHT projection values using about $3N/2$ additions per window. In the computation of partial projection values for sliding windows, the proposed algorithm requires only 1.5 additions per projection vector for each window. As shown by experimental results in Section VII, the computational time required by the proposed algorithm for computing ten or more projection values is about 75% of that of the GCK algorithm.

The rest of the paper is organized as follows. Section II defines terms and symbols used in this paper. Then the WHT algorithm in [9] is briefly introduced. In Section III, we introduce two examples of the proposed algorithm. Section IV illustrates the proposed algorithm for 1-D order-N WHT. The algorithm computes order-N WHT using order-4 and order-$N/4$ WHT. In Section V, the number of additions required by the proposed algorithm is derived. Section VI gives the experimental result of motion estimation in video coding application, which utilizes the proposed algorithm for computing 2-D WHT on sliding windows. Finally, Section VII presents conclusions.

II. Walsh Hadamard Transform on Sliding Windows

A. Definitions

Consider K input signal elements x_n where n=0, 1, …, K-1, which will be divided into overlapping windows of size N (K>N). Let the jth input window be:

$$\vec{X}_N(j) = [x_j, x_{j+1}, \ldots, x_{j+N-1}]^T \text{ for } j = 0, 1, \ldots, K-N. \quad (1)$$

A 1-D order-N WHT transforms N numbers into N projection values. Let M_N be an order-N WHT matrix and

$$M_N = [\vec{M}_N(0), \vec{M}_N(1), \ldots, \vec{M}_N(N-1)]^T \text{ where } \vec{M}_N(i) \text{ is the } i\text{th WHT basis vector.} \quad (2)$$

Let $y_N(i, j)$ be the ith WHT projection value for the jth window and

$$y_N(i,j)=\vec{M}_N(i)^T \vec{X}_N(j) \text{ for } i = 0, 1, \ldots N\text{-}1; j = 0, 1, \ldots, K\text{-}N. \tag{3}$$

In [10], $\vec{M}_N(i)$ and $y_N(i, j)$ are called the ith projection kernel and projection result respectively. Let $\vec{Y}_N(j)$ be the projection vector containing all projection values of the jth window and

$$\vec{Y}_N(j)=[y_N(0, j), y_N(1, j), \ldots, y_N(N\text{-}1, j)]^T= M_N \vec{X}_N(j). \tag{4}$$

For example, when N=4, we have:

$$\vec{Y}_4(j) = \begin{pmatrix} y_4(0,j) \\ y_4(1,j) \\ y_4(2,j) \\ y_4(3,j) \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{pmatrix} x_j \\ x_{j+1} \\ x_{j+2} \\ x_{j+3} \end{pmatrix} = M_4 \vec{X}_4(j). \tag{5}$$

The WHT in (5) is in sequency order. Its basis vectors are ordered according to the number of zero crossings. The relationship between WHTs in sequency order, dyadic order and natural order can be found in [15].

B. Previous WHT computation methods

In [9], Hel-Or and Hel-Or proposed a fast algorithm that computes $\vec{Y}_N(j)$ from $\vec{Y}_{N/2}(j)$ and $\vec{Y}_{N/2}(j+N/2)$ using (6).

$$y_N(i,j) = \begin{cases} y_{N/2}(\lfloor i/2 \rfloor, j) + y_{N/2}(\lfloor i/2 \rfloor, j+N/2), i\%4 = 0 \;\; or \;\; 3 \\ y_{N/2}(\lfloor i/2 \rfloor, j) - y_{N/2}(\lfloor i/2 \rfloor, j+N/2), i\%4 = 1 \;\; or \;\; 2 \end{cases}, \tag{6}$$

where % is the module operation; and $\lfloor \cdot \rfloor$ is the floor function.

As shown in Fig. 1, their algorithm first computes WHT projection values for window size N being 2, which are then used to compute WHT projection values for window size being 4 and so on. The computation starts at the root and moves down the tree until the projection values represented by the leaves are computed. The algorithm in [9] requires 1 addition per window along each node of the tree in Fig. 1. The GCK algorithm [10] utilizes previously computed order-N projection values for computing the current order-N projection value. When a small number of projection values are computed, the GCK algorithm requires 2 additions per window for each projection value while the algorithm in [9] requires $O(\log N)$ additions. When all projection values are computed, both the algorithms in [9] and [10] requires about $2N$ additions.

A new fast algorithm which is more efficient than that reported in [9][10], is proposed in this paper. It can efficiently compute order-N WHT on sliding windows, i.e. $y_N(i, j)$ for $i = 0, \ldots, P\text{-}1$ ($P \le N$) and $j = 0, 1, 2, \ldots, K\text{-}N$.

III. Fast Algorithm for WHT on Sliding Windows for Window Sizes 4 and 8

This section gives examples of computing order-N WHT on sliding windows of sizes N=4 and 8 using the proposed algorithm.

A. *Fast Algorithm for Window Size 4*

The proposed algorithm and the GCK algorithm [10] for window size being 4 are described in Table I. The proposed algorithm computes $\vec{Y}_4(j+1)$ (the WHT projection values in window j+1) using $\vec{Y}_4(j)$ (the computed projection values in window j) as shown in Table I. Except for the 0th projection value $y_4(0, j+1)$, the GCK algorithm utilizes the previous order-4 projection values to compute the current order-4 projection value.

TABLE I

FAST ALGORITHM WHEN WINDOW SIZE IS 4

	x_j	x_{j+1}	x_{j+2}	x_{j+3}	x_{j+4}	Proposed algorithm	GCK algorithm[10]
$y_4(0, j)$	1	1	1	1		$y_4(0, j+1)$ $= y_4(0, j) - x_j + x_{j+4}$	$y_4(0, j+1)$ $= y_4(0, j) - x_j + x_{j+4}$
$y_4(0, j+1)$		1	1	1	1		
$y_4(2, j)$	1	-1	-1	1		$y_4(1, j+1)$ $=-y_4(2, j)+x_j-x_{j+4}$	$y_4(1, j+2)$ $= y_4(0, j)-y_4(0, j+2)-y_4(1, j)$
$y_4(1, j+1)$		1	1	-1	-1		
$y_4(1, j)$	1	1	-1	-1		$y_4(2, j+1)$ $= y_4(1, j)-x_j+x_{j+4}$	$y_4(2, j+2)$ $= y_4(1, j+1)-y_4(1, j+2)-y_4(2, j+1)$
$y_4(2, j+1)$		1	-1	-1	1		
$y_4(3, j)$	1	-1	1	-1		$y_4(3, j+1)$ $= -y_4(3, j)+ x_j-x_{j+4}$	$y_4(3, j+2)$ $= y_4(3, j) -y_4(2, j)-y_4(2, j+2)$
$y_4(3, j+1)$		1	-1	1	-1		

Define $d_N(j)$ as:

$$d_N(j) = x_j - x_{j+N} . \tag{7}$$

We can see from Table I that the WHT projection values in window j+1 can be computed from those in window j and $d_N(j)$. Therefore, we have

$$\begin{bmatrix} y_4(0, j+2) \\ y_4(1, j+2) \\ y_4(2, j+2) \\ y_4(3, j+2) \end{bmatrix} = \begin{bmatrix} y_4(0, j) \\ -y_4(2, j) \\ y_4(1, j) \\ -y_4(3, j) \end{bmatrix} + \begin{bmatrix} -d_4(j) \\ d_4(j) \\ -d_4(j) \\ d_4(j) \end{bmatrix}, \tag{8}$$

and Fig. 2 shows the signal flow diagram.

Thus, after obtaining $\vec{Y}_4(j)$, the proposed algorithm obtain $\vec{Y}_4(j+1)$ by 5 additions as shown in (7)-(8) whereas the algorithm in [9] requires 6 additions and the GCK algorithm requires 8 additions.

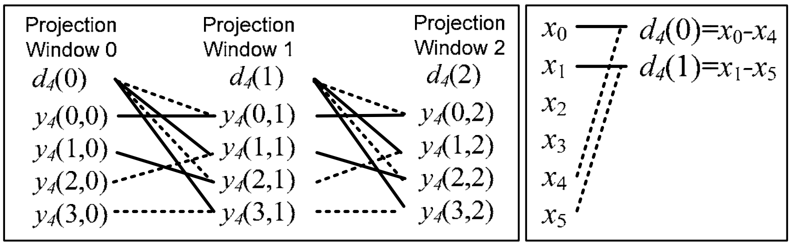


Fig. 2 Signal flow diagram of the Bottom up algorithm for window size equal 4

B. *Fast Algorithm for Window Size 8*

The proposed algorithm and the GCK algorithm [10] for window size being 8 are described in Table II. The proposed algorithm computes $\vec{Y}_8(j+2)=[y_8(0,j+2), y_8(1,j+2), \ldots, y_8(7,j+2)]^T$, which is the WHT projection vector in window j+2 for j=0, 1, …K-8, using $\vec{Y}_8(j)$, which is the computed projection vector in window j.

TABLE II

FAST ALGORITHM WHEN WINDOW SIZE IS 8

	x_j	x_{j+1}	x_{j+2}	x_{j+3}	x_{j+4}	x_{j+5}	x_{j+6}	x_{j+7}	x_{j+8}	x_{j+9}	Proposed algorithm	GCK
$y_8(0, j)$	1	1	1	1	1	1	1	1			$y_8(0,j+2)$	$y_8(0,j+2)$
$y_8(0, j+2)$			1	1	1	1	1	1	1	1	$=y_8(0,j)-x_j-x_{j+1}+x_{j+8}+x_{j+9}$	$=y_8(0,j+1)-x_{j+1}+x_{j+9}$
$y_8(2, j)$	1	1	-1	-1	-1	-1	1	1			$y_8(1,j+2)$	$y_8(1, j+2)$
$y_8(1, j+2)$			1	1	1	1	-1	-1	-1	-1	$=-y_8(2,j)+x_j+x_{j+1}-x_{j+8}-x_{j+9}$	$= y_8(0, j-2)-y_8(0, j+2)-y_8(1, j-2)$
$y_8(1, j)$	1	1	1	1	-1	-1	-1	-1			$y_8(2,j+2)$	$y_8(2, j+2)$
$y_8(2, j+2)$			1	1	-1	-1	-1	-1	1	1	$=y_8(1,j)-x_j-x_{j+1}+x_{j+8}+x_{j+9}$	$= y_8(1, j)-y_8(1, j+2)-y_8(2, j)$
$y_8(3, j)$	1	1	-1	-1	1	1	-1	-1			$y_8(3,j+2)$	$y_8(3, j+2)$
$y_8(3, j+2)$			1	1	-1	-1	1	1	-1	-1	$=-y_8(3,0)+x_j+x_{j+1}-x_{j+8}-x_{j+9}$	$=y_8(3, j-2)-y_8(2, j+2)-y_8(2, j-2)$
$y_8(4, j)$	1	-1	-1	1	1	-1	-1	1			$y_8(4,j+2)$	$y_8(4, j+2)$
$y_8(4, j+2)$			1	-1	-1	1	1	-1	-1	1	$=-y_8(4,j)+x_j-x_{j+1}-x_{j+8}+x_{j+9}$	$= y_8(3, j+1)-y_8(3, j+2)-y_8(4, j+1)$
$y_8(6, j)$	1	-1	1	-1	-1	1	-1	1			$y_8(5,j+2)$	$y_8(5, j+2)$
$y_8(5, j+2)$			1	-1	-1	1	-1	1	1	-1	$=y_8(6,j)-x_j+x_{j+1}+x_{j+8}-x_{j+9}$	$= y_8(4, j-2)-y_8(4, j+2)-y_8(5, j-2)$
$y_8(5, j)$	1	-1	-1	1	-1	1	1	-1			$y_8(6,j+2)$	$y_8(6, j+2)$
$y_8(6, j+2)$			1	-1	1	-1	-1	1	-1	1	$=-y_8(5,j)+x_j-x_{j+1}-x_{j+8}+x_{j+9}$	$= y_8(6, j) -y_8(5, j)-y_8(5, j+2)$
$y_8(7, j)$	1	-1	1	-1	1	-1	1	-1			$y_8(7,j+2)$	$y_8(7, j+2)$
$y_8(7, j+2)$			1	-1	1	-1	1	-1	1	-1	$=y_8(7,j)-x_j+x_{j+1}+x_{j+8}-x_{j+9}$	$= y_8(7, j-2) -y_8(6, j-2)-y_8(6, j+2)$

Define $t_{N/4}(i, j)$ as:

$$t_{N/4}(i, j) = y_{N/4}(i, j) - y_{N/4}(i, j+N) \text{ for } i = 0, \ldots, N/4-1;\ j = 0, 1, \ldots, K-5N/4. \quad (9)$$

For N=8, we have:

$$\begin{bmatrix} t_2(0,j) \\ t_2(1,j) \end{bmatrix} = \begin{bmatrix} y_2(0,j) \\ y_2(1,j) \end{bmatrix} - \begin{bmatrix} y_2(0,j+8) \\ y_2(1,j+8) \end{bmatrix}.$$

From (3) and then (7), we have:

$$\begin{aligned} \begin{bmatrix} t_2(0,j) \\ t_2(1,j) \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_j \\ x_{j+1} \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_{j+8} \\ x_{j+9} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_j - x_{j+8} \\ x_{j+1} - x_{j+9} \end{bmatrix} = M_2 \begin{bmatrix} d_8(j) \\ d_8(j+1) \end{bmatrix}. \end{aligned} \quad (10)$$

As given by (10), $t_2(i, j)$ is the ith order-2 WHT projection value of $[d_8(j), d_8(j+1)]^T$. According to Table II and (10), WHT projection vector in window j+2 can be computed from those in window j as well as $t_{N/4}(i, j)$ as follows:

$$\begin{bmatrix} y_8(0,j+2) \\ y_8(1,j+2) \\ y_8(2,j+2) \\ y_8(3,j+2) \\ y_8(4,j+2) \\ y_8(5,j+2) \\ y_8(6,j+2) \\ y_8(7,j+2) \end{bmatrix} = \begin{bmatrix} y_8(0,j) \\ -y_8(2,j) \\ y_8(1,j) \\ -y_8(3,j) \\ -y_8(4,j) \\ y_8(6,j) \\ -y_8(5,j) \\ y_8(7,j) \end{bmatrix} + \begin{bmatrix} -t_2(0,j) \\ t_2(0,j) \\ -t_2(0,j) \\ t_2(0,j) \\ t_2(1,j) \\ -t_2(1,j) \\ t_2(1,j) \\ -t_2(1,j) \end{bmatrix}. \quad (11)$$

In summary, (11) can be represented by:

$$y_8(i,j+2) = \begin{cases} (-1)^{v+i}\{y_8(i,j) - t_2(v,j)\}, i = 0,3,4,7 \\ (-1)^{v+i}\{y_8[i-(-1)^i, j] - t_2(v,j)\}, i = 1,2,5,6 \end{cases}, \text{ where } v = \lfloor i/4 \rfloor. \quad (12)$$

TABLE III

COMPUTATION OF ALL ORDER-8 WHT PROJECTION VALUES IN WINDOW J+2

Step a $d_8(j+1) = x_{j+1}-x_{j+9}$.
- One addition is required.
Step b $t_2(0,j) = [1, 1]\, [d_8(j), d_8(j+1)]^T$ and $t_2(1,j) = [1, -1]\, [d_8(j), d_8(j+1)]^T$.
- Two additions are required. Note that $d_8(j)$ was obtained during computation of $\vec{Y}_8(j+1)$ in *Step a*.
Step c $y_8(i,j+2) = \begin{cases} (-1)^{v+i}\{y_8(i,j) - t_2(v,j)\}, i = 0,3,4,7 \\ (-1)^{v+i}\{y_8[i-(-1)^i, j] - t_2(v,j)\}, i = 1,2,5,6 \end{cases}$, where $v = \lfloor i/4 \rfloor$.
- Eight additions are required in this step for i = 0, 1, …, 7.

Table III gives the three steps for computing $\vec{Y}_8(j+2)$ from $\vec{Y}_8(j)$ as well as $\vec{Y}_8(j+1)$ and the corresponding number of operations required.

Therefore, the proposed algorithm requires 11 additions whereas the algorithm in [9] requires 14 additions for obtaining the 8 projection values in $\vec{Y}_8(j+2)$. The GCK algorithm requires 16 additions.

IV. FAST ALGORITHM FOR WHT ON SLIDING WINDOWS FOR WINDOW SIZE *N*

A. The algorithm

Let D_N be the order-N reverse-identity matrix, i.e., elements at the reverse-diagonal positions are 1 and 0 at others. For example, D_4 is:

$$D_4 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

The equation below is proved in the appendix:

$$\begin{bmatrix} y_N(8i+0, j) \\ y_N(8i+1, j) \\ y_N(8i+2, j) \\ y_N(8i+3, j) \\ y_N(8i+4, j) \\ y_N(8i+5, j) \\ y_N(8i+6, j) \\ y_N(8i+7, j) \end{bmatrix} = \begin{bmatrix} M_4 & \\ & D_4 M_4 \end{bmatrix} \begin{bmatrix} y_{N/4}(2i, j) \\ y_{N/4}(2i, j+N/4) \\ y_{N/4}(2i, j+2N/4) \\ y_{N/4}(2i, j+3N/4) \\ y_{N/4}(2i+1, j) \\ y_{N/4}(2i+1, j+N/4) \\ y_{N/4}(2i+1, j+2N/4) \\ y_{N/4}(2i+1, j+3N/4) \end{bmatrix} \quad \text{for } i=0, 1, \ldots, N/8-1. \tag{13}$$

Hence, order-N WHT is partitioned into $N/4$ groups of order-4 WHT. Utilizing the method in (8) for each of the order-4 WHT in (13), WHT projection values in window $j+N/4$ can be computed using projection values in window j as well as $t_{N/4}(2i, j)$ and $t_{N/4}(2i+1, j)$ as follows:

$$\begin{bmatrix} y_N(8i+0, j+N/4) \\ y_n(8i+1, j+N/4) \\ y_N(8i+2, j+N/4) \\ y_N(8i+3, j+N/4) \\ y_N(8i+4, j+N/4) \\ y_N(8i+5, j+N/4) \\ y_N(8i+6, j+N/4) \\ y_N(8i+7, j+N/4) \end{bmatrix} = \begin{bmatrix} y_N(8i+0, j) \\ -y_N(8i+2, j) \\ y_N(8i+1, j) \\ -y_N(8i+3, j) \\ -y_N(8i+4, j) \\ y_N(8i+6, j) \\ -y_N(8i+5, j) \\ y_N(8i+7, j) \end{bmatrix} + \begin{bmatrix} -t_{N/4}(2i+0, j) \\ t_{N/4}(2i+0, j) \\ -t_{N/4}(2i+0, j) \\ t_{N/4}(2i+0, j) \\ t_{N/4}(2i+1, j) \\ -t_{N/4}(2i+1, j) \\ t_{N/4}(2i+1, j) \\ -t_{N/4}(2i+1, j) \end{bmatrix}, \tag{14}$$

where $t_{N/4}(i, j)$ is defined in (9). Equation (14), which is for order-N WHT, becomes (11) when N=8. Let us first consider the computation of $t_{N/4}(i, j)$ in (14). Utilizing the $d_N(j)$ in (7), we define:

$$\vec{D}_{N/4}(j)=[d_N(j),\, d_N(j+1),\, \ldots,\, d_N(j+N/4-1)]^{\mathrm{T}}=[\vec{X}_{N/4}(j)-\vec{X}_{N/4}(j+N)]. \tag{15}$$

From (3), (9) and then (15), we have:

$$\begin{aligned} t_{N/4}(i,j) &= y_{N/4}(i,j) - y_{N/4}(i,j+N) \\ &= \vec{M}_{N/4}(i)^{\mathrm{T}}\vec{X}_{N/4}(j) - \vec{M}_{N/4}(i)^{\mathrm{T}}\vec{X}_{N/4}(j+N) = \vec{M}_{N/4}(i)^{\mathrm{T}}[\vec{X}_{N/4}(j) - \vec{X}_{N/4}(j+N)] \\ &= \vec{M}_{N/4}(i)^{\mathrm{T}}\vec{D}_{N/4}(j). \end{aligned} \tag{16}$$

Equation (16) shows that $t_{N/4}(i, j)$ is the ith projection value of the order-$N/4$ WHT of $\vec{D}_{N/4}(j)$. When N=8, (16) becomes (10).

The signal flow diagram in Fig. 3 depicts the computation of order-N WHT using (14)-(16). Table IV describes the computation of $\vec{Y}_N(j+N/4)$ from $\vec{Y}_N(j)$ and the corresponding number of operations as well as memory required. Since at most size $2K$ memory is required for the proposed algorithm at each step, the memory required for the proposed algorithm is $2K$ which is the same as the GCK algorithm.

This paper focuses on 1-D WHT. However, it is easy to extend the proposed 1-D WHT algorithm to higher dimensions. For example, when the 2-D WHT of size $N\times N$ is computed, our algorithm in Table IV can use GCK for computing the WHT of size $N\times N/4$ in *Step b* and using the *Step c* to obtain the projection values of size $N\times N$. In this way, we require 1 addition in *Step a*, $N^2/2$ additions in *Step b* and N^2 additions in *Step c*, i.e. $1.5N^2$ additions in all. In this way, the proposed algorithm requires 1.5 additions per window per projection value independent of dimension. In Section VI, we will show the experimental result that uses the fast algorithm for 2-D WHT.

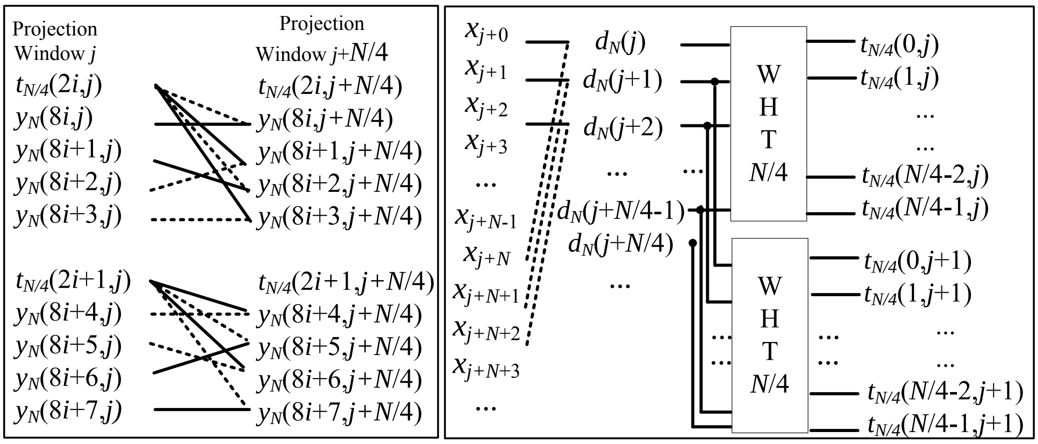


Fig. 3 Signal flow diagram of the bottom up algorithm for order-N sequency WHT

TABLE IV

COMPUTATION OF ORDER-N WHT

Overall procedure:

For each j {*Step a*};

For each i

{

For each j { *Step b*; }

For each j { *Step c*; }

}

Step a Compute $d_N(j) = x_j - x_{j+N}$. This step provides the $\vec{D}_{N/4}(j)$ in (16).

Analysis: One addition per window is required. Size $2K$ memory is required for storing $d_N(j)$ and the input data x_j for j=0,… K-1.

Step b Compute $t_{N/4}(\lfloor i/4 \rfloor, j) = \vec{M}_{N/4}(\lfloor i/4 \rfloor)^T \vec{D}_{N/4}(j)$. This step provides the $t_{N/4}$ in (14). We can use the GCK algorithm in [10] for computation.

Analysis: $N/2$ additions per window are required for the $N/4$ values of $t_{N/4}(\lfloor i/4 \rfloor, j)$ for given j. As stated in [10], size $2K$ memory is required by GCK.

Step c Obtain $y_N(i,j+N/4)$ using (14). Note that the $y_N(i,j)$ in (14) is computed previously.

Analysis: N additions per window are required for the N values of i. Size K memory is required for storing the $t_{N/4}(i, j)$ for j=0,… K-1 which are computed in *Step b*; size $O(N)$ memory is required for storing at most two projection values $y_{N/4}(i, a)$ for $j \le a < j+N/4$ required in the right hand side of (14) for the given i. Since we have $N \ll K$ for most cases, the memory requirement is less than $2K$ in this step.

V. COMPUTATION REQUIREMENT OF THE PROPOSED FAST ALGORITHMS FOR WINDOW SIZE N

A. *When all projection values are computed*

Let the total number of additions for obtaining $\vec{Y}_N(j+N/4)$ be $B_N(N)$. According to the analysis in Table IV, we have:

$$B_N(N) = 1 + N/2 + N = 3N/2 + 1.$$

The number of additions required for the GCK algorithm and the proposed algorithm are summarized in Table V which shows that the proposed algorithm requires about $3N/2$ additions while the GCK algorithm requires $2N$ additions. The number of additions required by our algorithm for order-4 and order-8 WHT are 5 and 11 respectively because we can use direct computation instead of the GCK for calculating $t_{N/4}(i, j)$ in the *Step b* of Table IV. For example, if N=4, then $t_{N/4}(i, j)=d_4(j)$ and no computation is required in the *Step b* of Table IV for obtaining $t_{N/4}(i, j)$.

TABLE V

NUMBERS OF ADDITIONS REQUIRED BY THE GCK ALGORITHM AND THE PROPOSED ALGORITHM FOR ALL PROJECTION VALUES OF ORDER-N WHT

Size	4	8	16	32	N
GCK	8	16	32	64	$2N$
Proposed	5	11	25	49	$3N/2+1$

B. *When not all projection values are computed*

In many applications, not all projection values are required. In this part, we analyze the computational requirement when only the first P projection values are computed for window size N. Specifically, we shall derive the number of additions for the computation of $y_N(0,j)$, $y_N(1, j)$, … and $y_N(P\text{-}1, j)$ for $j = N/4$, $N/4+1$, …, $K\text{-}N$. Here we shall not consider the cases when $j<N/4$ because the computational complexity is negligible as $N << K$ in most cases. A zero padding approach dealing with the cases when $j<N/4$ is introduced in [8].

TABLE VI

COMPUTATION OF ORDER-N WHT WHEN NOT ALL PROJECTION VALUES ARE REQUIRED

The overall procedure and the three steps are the same as that in Table IV. The only difference is that the total number of i is P now.

***Analysis*:**

***Step a*:** 1 addition per window is required in this step.

***Step b*:** $2\cdot\lceil P/4\rceil$ additions are required in this step using the GCK for the $\lceil P/4\rceil$ values of $t_{N/4}(\lfloor i/4\rfloor, j)$.

***Step c*:** If P % 4 ≡ 2 (for example P is 6 or 10), for the computation in (14), the proposed algorithm need to compute $y_N(4\cdot\lfloor P/4\rfloor+2,\ j)$ for $y_N(4\cdot\lfloor P/4\rfloor+1,\ j+N/4)$. So P+1 additions are required if the P % 4 ≡ 1; Otherwise, P additions are required.

Let the number of additions per window required to obtain $y_N(i,\ j+N/4)$ for i=0,…, P-1; j=0, 1, …, K-5N/4 be $B_N(P)$. Table VI lists the steps and the corresponding number of additions required. As shown in Table VI, we require 1 addition in *Step a*, $2\cdot\lceil P/4\rceil$ additions in *Step b* and at most P+1 additions in *Step c*. The number of additions required for obtaining P projection values in order-N WHT using the proposed algorithm as given in Table VI has the following inequality:

$$B_N(P) \le 1+2\cdot\lceil P/4\rceil+P+1 \le \lceil 3P/2\rceil+3. \qquad (18)$$

The computation required is about 1.5 additions/pixel/kernel using the proposed algorithm.

VI. Experimental Results

To investigate the computational efficiency of the proposed algorithm for pattern matching in practical applications, block matching in motion estimation is utilized. Block matching in motion estimation using fast WHT was carried out on the first 200 frames of a video sequence "tempete" which has a resolution of 352×288. The experiment considers the execution time required for obtaining different numbers of WHT projection values, which ranges from 1 to 20. The proposed algorithm is compared with the algorithm in [8], which utilized the GCK algorithm.

In a similar experiment reported in [8], two projection value computation orders were used. They are the "snake order" and "increasing frequency order". Fig. 4 shows the ordering of the first 20 projection values of these two orders. The percentage of the time required by the proposed algorithm with respect to the GCK algorithm is given in Fig. 6. The proposed algorithm outperforms the GCK algorithm when the number of projections is greater than 6. As the proposed algorithm computes 3 or 4 projection values together to save computation whereas the GCK algorithm does not, so the percentage of computational time saved by the proposed algorithm in comparison with the GCK algorithm depends on the number of projections. Generally, the proposed algorithm achieves a higher saving when most projection values to be computed can take advantage of this property. This is why when the number of projection values approaches 13 and 16 for snake order, the proposed algorithm requires the least percentage of time compared with the GCK algorithm. When the number of projection values is less than 5, the proposed algorithm requires more computational time because projection values cannot be grouped together for computation. Therefore, we would suggest the use of the GCK algorithm when the number of projection values is less than 5.

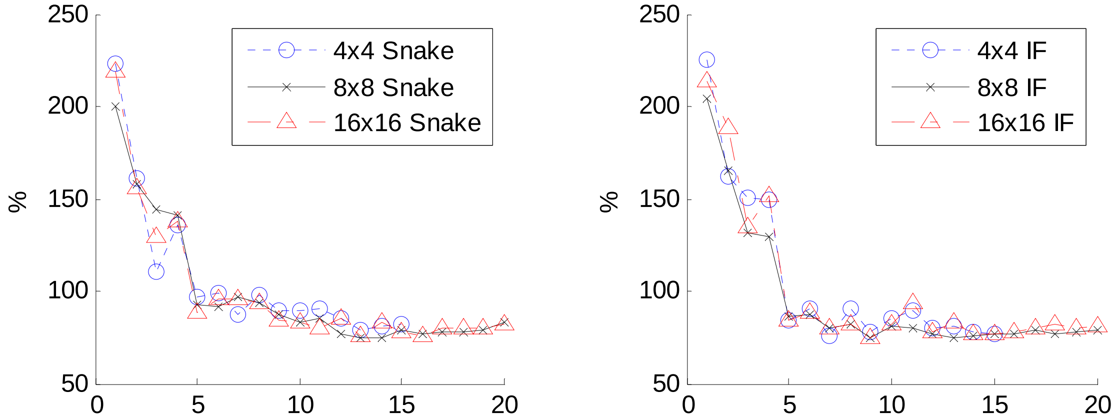
0	1	8	9	
3	2	7	10	
4	5	6	11	
15	14	13	12	
16	17	18	19	

(a) Snake order

0	2	5	10	16
1	3	7	12	18
4	6	9	14	
8	11	13	19	
15	17			

(b) Increasing frequency order

Fig. 4 Two different projection orders



(a) Snake order (b) Increasing frequency order

Fig. 5 The percentage of time required by our algorithm with respect to GCK algorithm, where Snake stands for the snake order and IF stands for the increasing frequency order. The experiment is implemented on a 2.13GHz PC using C on windows XP system with compiling environment VC 6.0.

VII. Conclusions

This paper proposes a fast computational algorithm for Walsh Hadamard Transform on sliding windows, which requires about 1.5 additions per projection vector per window. The computational time of the proposed algorithm is about 75% that of the GCK algorithm which is the fastest algorithm reported so far. In cases where not all projection values are needed, the proposed algorithm can outperform the GCK algorithm when the number of projection values is five or above. The proposed algorithm achieves its high efficiency in the computation of order-N WHT by using order-4 and order-N/4 WHT. This paper provides fast algorithm for 1-D WHT. In the future, we are going to seek even faster algorithm. We will also try to see if there exists the superset of GCK that can be computed by constant number of additions per window per projection value independent of the size and dimension of the transform.

ACKNOWLEDGEMENT

The work described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China CUHK410307. The authors are also thankful to Prof. Hel-Ors' for providing their code implementing the GCK algorithm.

APPENDIX A

This appendix provides the proof for (13). Except for this appendix, sequency order is used for representing WHT. In this appendix, dyadic-ordered WHT will be utilized for proving (13). Natural order-N WHT can be represented by:

$$M_N = M_2 \otimes M_{N/2},$$

where $\otimes$ is the Kronecker product ($A \otimes B$ is a $mp \times nq$ matrix composed of the $m \times n$ blocks $(a_{i,j} B)$) and

$$M_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Both sequency and dyadic orders [15] are the reordering form of the natural order for WHT. Here we denote M_N^Z as the order-N sequency-ordered WHT matrix; denote M_N^D as the order-N dyadic-ordered WHT matrix and:

$$M_N^D = [\vec{M}_N^D(0),\ \vec{M}_N^D(1),\ \ldots,\ \vec{M}_N^D(N\text{-}1)]^T \text{ where } \vec{M}_N^D(i) \text{ is the } i\text{th WHT basis vector.} \quad \text{(a1)}$$

The binary vector representation of i in (a1) $[i_1, i_2, \ldots, i_g]^T$, where i_k are 0 or 1 for $k = 1, \ldots g$ and:

$$i = 2^{g-1} i_1 + 2^{g-2} i_2 + \ldots + 2 i_{g-1} + i_g.$$

For dyadic-ordered WHT, for b=0,…,a-1, a= 2, 4, 8, …,N, we have:

$$\vec{M}_N^D(ai+b) = \vec{M}_a^D(b) \otimes \vec{M}_{N/a}^D(i). \quad \text{(a2)}$$

Let i^Z and i^D be index of sequency-ordered and dyadic-ordered WHT respectively.

As pointed out in [15], the relationship between the binary vector representation of i^Z and i^D is:

$$i^Z = [W_{D,Z}]_g\, i^D, \quad \text{(a3)}$$

where $$[W_{D,Z}]_g=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 1 & 1 & \ldots & 1 \end{bmatrix}_{g\times g}.$$

According to (a3), if $\vec{M}_N^Z(ai^Z+b_i^Z)=\vec{M}_N^D(ai^D+b^D)$, where b^Z, $b^D<a$, $a=2^k$, then i^Z is only decided by i^D. So we have:

$$\vec{M}_N^Z(ai^Z+b_i^Z)=\vec{M}_N^D(ai^D+b^D) \;=>\; \vec{M}_{N/a}^Z(i^Z)=\vec{M}_{N/a}^D(i^D)\,. \tag{a4}$$

Denote $f(b,a,i)$ as:

$$f(b,a,i)=\begin{cases} b, & i \text{ is an even number} \\ a-1-b, & i \text{ is an odd number} \end{cases}, \tag{a5}$$

The b_i^Z in (a4) is decided by both i^Z and b^D:

$$b_i^Z=f([W_{D,Z}]b^D,a,i^Z),$$

where the size of $[W_{D,Z}]$ is $\log_2 a\times\log_2 a$.

It is obvious that $f[f(b,a,i),a,i]=b$, so we have:

$$[W_{D,Z}]b^D=f[\,f([W_{D,Z}]b^D,a,i^Z),a,i^Z]=f(b_i^Z,a,i^Z). \tag{a6}$$

$\vec{M}_N^Z(ai^Z+b_i^Z,j)$ can be represented as follows using (a2), (a4) and (a6):

$$\begin{aligned}\vec{M}_N^Z(ai^Z+b_i^Z,j)&=\vec{M}_N^D(ai^D+b^D,j)=\vec{M}_a^D(b^D)\otimes\vec{M}_{N/a}^D(i^D)\\ &=\vec{M}_a^Z([W_{D,Z}]b^D)\otimes\vec{M}_{N/a}^Z(i^Z)=\vec{M}_a^Z[f(b_i^Z,a,i^Z)]\otimes\vec{M}_{N/a}^Z(i^Z)\end{aligned}. \tag{a7}$$

According to (a7), we have:

$$\vec{M}_N^Z(ai+b,j)=\vec{M}_a^Z[f(b,a,i)]\otimes\vec{M}_{N/a}^Z[i]. \tag{a8}$$

Therefore, $y_N^Z(ai+b,j)$ can be represented as follows:

$$\begin{aligned} y_N^Z(ai+b,j)&=\{\vec{M}_a^Z[f(b,i,a)]\otimes\vec{M}_{N/a}^Z[i]\}\vec{X}_N\\ &=[\vec{M}_a^z[f(b,i,a)]I_a]\otimes[I_1\vec{M}_{N/a}^z(i)]\vec{X}_N\\ &=[I_1\otimes\vec{M}_a^z[f(b,i,a)]][I_a\otimes\vec{M}_{N/a}^z(i)]\vec{X}_N\\ &=\vec{M}_a^Z[f(b,i,a)]\begin{bmatrix}\vec{M}_{N/a}^z(i) & & & \\ & \vec{M}_{N/a}^z(i) & & \\ & & \ldots & \\ & & & \vec{M}_{N/a}^z(i)\end{bmatrix}\vec{X}_N\\ &=M_a^Z[f(b,a,i)]\begin{bmatrix} y_{N/a}^Z(i,j)\\ y_{N/a}^Z(i,j+N/a)\\ \ldots\\ y_{N/a}^Z(i,j+N-N/a)\end{bmatrix}_{a\times 1}\end{aligned}. \tag{a9}$$

The following equation is valid using (a9):

$$\begin{bmatrix} y_N^Z(2ai^Z+0,j) \\ y_N^Z(2ai^Z+1,j) \\ \cdots \\ y_N^Z(2ai^Z+a-1,j) \\ y_N^Z(2ai^Z+a,j) \\ y_N^Z(2ai^Z+a+1,j) \\ \cdots \\ y_N^Z(2ai^Z+2a-1,j) \end{bmatrix} = \begin{bmatrix} M_a^Z \begin{pmatrix} y_{N/a}^Z(2i^Z,j) \\ y_{N/a}^Z(2i^Z,j+N/a) \\ \cdots \\ y_{N/a}^Z(2i^Z,j+N-N/a) \end{pmatrix} \\ D_a M_a^Z \begin{pmatrix} y_{N/a}^Z(2i^Z+1,j) \\ y_{N/a}^Z(2i^Z+1,j+N/a) \\ \cdots \\ y_{N/a}^Z(2i^Z+1,j+N-N/a) \end{pmatrix} \end{bmatrix} = \begin{bmatrix} M_a^Z & \\ & D_a M_a^Z \end{bmatrix} \begin{bmatrix} y_{N/a}^Z(2i^Z,j) \\ y_{N/a}^Z(2i^Z,j+N/a) \\ \cdots \\ y_{N/a}^Z(2i^Z,j+N-N/a) \\ y_{N/a}^Z(2i^Z+1,j) \\ y_{N/a}^Z(2i^Z+1,j+N/a) \\ \cdots \\ y_{N/a}^Z(2i^Z+1,j+N-N/a) \end{bmatrix}. \tag{a10}$$

Equ. (13) is valid when a=4 in (a10).

REFERENCES

[1] M. S. Aksoy , O. Torkul, and I. H. Cedimoglu, "An industrial visual inspection system that uses inductive learning," *Journal of Intelligent Manufacturing*, vol. 15(4), pp. 569-574, Aug. 2004.

[2] A.W. Fitzgibbon, Y. Wexler, and A. Zisserman, "Image-Based rendering using image-based priors," *Proc. Int'l Conf. Computer Vision,* vol. 2, pp. 1176-1183, Oct. 2003.

[3] T. Luczak and W. Szpankowski, "A suboptimal lossy data compression based on approximate pattern matching," *IEEE Trans. Information Theory*, vol. 43, pp. 1439-1451, Sept. 1997.

[4] R. M. Dufour, E. L. Miller, and N. P. Galatsanos, "Template matching based object recognition with unknown geometric parameters," *IEEE Trans. Image Process.*, vol. 11, no. 12, pp. 1385–1396, Dec. 2002.

[5] C.M. Mak., C.K. Fong, and W.K. Cham, "Fast motion estimation for H.264/AVC in Walsh Hadamard domain," *IEEE Trans. Circuits Syst. Video Technol.*, accepted.

[6] ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, JVT-G050, March, 2003.

[7] C.M. Mak., C.K. Fong, W.K. Cham, "Fast motion estimation for H.264/AVC in Walsh Hadamard domain," *IEEE Trans. Circuits Syst. Video Technol*., 18(6):735 – 745, Jun. 2008.

[8] Y. Moshe and H. Hel-Or, "A Fast Block Motion Estimation Algorithm Using Gray Code Kernels", in *Proc. IEEE Symp. Signal Processing and Information Technology*, pp.185 - 190, Vancouver, Canada, Aug. 2006.

[9] Y. Hel-Or and H. Hel-Or, "Real time pattern matching using projection kernels," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 27, no. 9, pp. 1430-1445, Sept. 2005.

[10] G. Ben-Artzi, H. Hel-Or, and Y. Hel-Or, "The gray-code filter kernels," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 29, no. 3, pp.382 – 393, Mar. 2007.

[11] S. Omachiand and M. Omachi, "Fast template matching with polynomials," *IEEE Trans. Image Process.,* vol. 16, no. 8, pp.2139 – 2149, Aug. 2007.

[12] G. J. VanderBrug and A. Rosenfeld, “Two-stage template matching,” *IEEE Trans. Comput.,* vol. C-26, no. 4, pp. 384–393, Apr. 1977.

[13] M. Ben-Yehuda, L. Cadany, H. Hel-Or, and Y. Hel-Or, “Irregular Pattern Matching using Projection,” in *Proc. IEEE International Conference on Image Processing*, Genoa Italy, Sept. 2005.

[14] J.L. Shanks, “Computation of the Fast Walsh-Fourier Transform**,”** *IEEE Trans. Comput,* Vol. C-18, No. 5, pp.457 – 459, May 1969.

[15] W.K. Cham and R.J. Clarke, “Dyadic Symmetry and Walsh Matrices,” *IEE Proceedings*, Pt.F., Vol.134, No.2, pp.141-145, Apr. 1987.

Wanli Ouyang received the B.S. degree in computer science from Xiangtan University, Hunan, China, in 2003. He received the M.S. degree with the College of Computer Science and Technology, Beijing University of Technology, Beijing, China. He is now pursuing Ph.D in the Department of Electronic Engineering, The Chinese University of Hong Kong.

Wai-Kuen Cham graduated from The Chinese University of Hong Kong in 1979 in Electronics. He received his M.Sc. and Ph.D. degrees from Loughborough University of Technology, U.K., in 1980 and 1983 respectively. Since May 1985, he has been with the Department of Electronic Engineering, The Chinese University of Hong Kong. Prof. Cham is a Chartered Engineer and a senior member of IEEE.