

# Parallel Hardware-Software Architecture for computation of Discrete Wavelet Transform using the Recursive Merge Filtering algorithm

Piyush Jamkhandi, Amar Mukherjee, Kunal Mukherjee and Robert Franceschini

School of Electrical Engineering and Computer Science,  
University of Central Florida,  
Orlando, Florida,  
USA

E-mail: {piyush, amar, mukherje, rfrances}@cs.ucf.edu

**Abstract.** We present an FPGA -based parallel hardware-software architecture for the computation of the Discrete Wavelet Transform (DWT), using the Recursive Merge Filtering (RMF) algorithm. The DWT is built in a bottom-up fashion in  $\log N$  steps, successively building complete DWTs by “merging” two smaller DWTs and applying the wavelet filter to only the “smooth” or DC coefficient from the smaller DWTs. The main bottleneck of this algorithm is the data routing process, which can be reduced by separating the computations into two types to introduce parallelism. This is achieved by using a virtual mapping structure to map the input. The data routing bottleneck has been transformed into simple arithmetic computations on the mapping structure. Due to the use of the FPGA -RAM for the mapping structure, the total number of data accesses to the main memory are reduced. This architecture shows how data routing in this problem can be transformed into a series of index computations

## 1 Introduction

In this paper, we present a multi-threaded architecture for the computation of Discrete Wavelet Transform (DWT) using the Recursive Merge Filtering (RMF) algorithm. The RMF algorithm[1] overcomes the main disadvantage of conventional wavelet coders-decoders of not being able to generate the code for the image until the complete image has been transformed. The use of the RMF algorithm overcomes the performance, functionality and reliability drawbacks of the current DWT methods. This paper builds upon the RMF algorithm by introducing an efficient data routing technique, based on the transformation of data routing operations to arithmetic computations. We show how the architecture can be used to compute the DWT of an image bottom-up and then carry out hierarchical merging of the sub-blocks to obtain the wavelet transform. To implement this multi- threaded architecture we use the

XC6200 FPGA, which suits the given application ideally. The essential idea is to separate the actual computation for the transform from the data routing. This can be achieved by using virtual position mapping for each image position. Using virtual mapping, the data computation can be performed on the actual data inputs, while the data routing is performed on the virtual mapping. Using a virtual coordinate system to define the position of the data being routed, the data routing is transformed into a set of coordinate additions/subtractions. We give a set of generalized equations, which can be applied to any block of the input data to perform the wavelet transform of the entire image.

We first describe the RMF algorithm formally. This is followed by an analysis of the various sub-tasks to be carried out for the DWT using RMF. We then present the formal notation for the transformation from data routing to simple coordinate addition and subtraction. The analysis is further developed to define the multi-threaded architecture for computation of the DWT using RMF. An implementation strategy using FPGAs is then described. Results of the simulation of this architecture showing the reduction in the main memory accesses are finally presented.

## 2 Formal Description of the Recursive Merge Filtering Algorithm

In this section we briefly describe the Recursive Merge Filter (RMF) algorithm to compute the DWT for 1-dimensional. The RMF algorithm computes the DWT of a 1-dimensional array of length  $N$  (where  $N=2^k$  for some positive integer  $k$ ) in a bottom-up fashion, by successively “merging” two smaller DWTs (four in 2-D), and applying the wavelet filter to only the “smooth” or DC coefficients.

### 2.1 RMF Operator

We will first formally define our primitive, the RMF operator, *in terms of the array indices of two DWT arrays* being merged. This takes as inputs two DWTs,  $DWT_1$  and  $DWT_2$ , each of length  $2^k$ , and outputs a DWT of length  $2^{k+1}$ :

$$\begin{aligned}
 & RMF[DWT_1(0:2^k-1), DWT_2(0:2^k-1)] \\
 &= RMF[DWT_1(0:2^{k-1}-1), DWT_2(0:2^{k-1}-1)] \bullet DWT_1(2^{k-1}:2^k-1) \bullet DWT_2(2^{k-1}:2^k-1) \\
 &\quad \text{if } k > 0 \\
 &= RMF[DWT_1(0:0), DWT_2(0:0)] = h(DWT_1(0), DWT_2(0)) \bullet g(DWT_1(0), DWT_2(0)) \\
 &\quad \text{if } k = 0
 \end{aligned}$$

The RMF operator is defined recursively on sub-arrays of the original DWTs. The first half of  $DWT_1$  and  $DWT_2$  are recursively passed to the RMF operator, and the remaining coefficients of the two DWTs are concatenated (symbolized by ‘ $\bullet$ ’) at the

end, as shown. The recursion terminates when the length of the DWTs being merged becomes equal to one - at this point, the RMF uses the Haar filters  $h$  and  $g[2]$ , to generate the low pass and high pass coefficients.

### 3 DWT in terms of the RMF operator

A recursive notation for the discrete wavelet transform (DWT) of an array  $x(n)$  of length  $N=2^k$ , which directly leads to a recursive procedure to compute the DWT is given below.

$$DWT[x(0:2^k-1)] = RMF[DWT[x(0:2^{k-1}-1), DWT[x(2^{k-1}:2^k-1)]] \quad \text{if } k > 1$$

$$DWT[x(0:1)] = [h(x(0), x(1)), g(x(0), x(1))] \quad \text{if } k = 1$$

The recursion terminates when the length of the array becomes two. At this point, the Haar filters  $h$  and  $g$  are applied to generate the low pass and high pass coefficients. For proof of the equivalence of the RMF algorithm and the FWT algorithm, refer to Mukherjee et al.[1]

### 4 RMF Algorithm Computations and Data Shifting

For a 2D image, the RMF process begins by computing the 2x2 transform of the 2D input data. This is done by selecting a 2x2-block row wise and computing its transform. This process is continued until all the 2x2 blocks in the image are transformed. Once all the blocks are transformed, sets of four adjacent 2x2 blocks are selected for the merging process. The process of merging the 2x2 blocks into 4x4 blocks continues until all the 2x2 blocks are merged into a set of 4x4 blocks. After the completion of this process, a set of four adjacent 4x4 blocks is merged to obtain the next level 8x8 block. This process of merging continues until all the blocks are merged to obtain a single merged block, at which time we are done with the process. A detailed extension of the RMF for 2D is given in Kunal et al. The merging process is inherently an exchange of blocks of data with the adjacent same-sized blocks representing four DWTs of four sub-images. If the size of the four quadrants of the merged data block is 2x2 then we apply the basic RMF-2D operation to the top-left matrix quadrant. In case the size of the quadrants is greater than 2x2 we recursively apply the *Merge* operation, until the size of the quadrants becomes 2x2 at which time we apply the RMF-2D operator to the top left quadrant. Along with the recursive *Merge* operations and the final RMF-2D operation, after a set of four blocks have been merged, we apply a 1D RMF on the bottom left quadrant in row wise manner. The 1D RMF operator is also applied to the upper right quadrant in column wise manner. One of the main disadvantages of the RMF algorithm, in its current form, is the large amount of data movement that has to be performed at every merge step.

## 5 Transformation of Data Routing to Address Computation

As seen above, the data shifting constitutes a major part of the overall process of DWT computation using RMF. We propose the use of a virtual mapping index called the *rMap* to reduce the total number of data routings. The *rMap* index is a pointer from every image position to a data value. The *rMap* index can be visualized as a structure with two components: x and y. The x and y values store the current co-ordinate position of a pixel on the transformed image. Consider a position (i, j) on an image. During the computation of the DWT the pixel value at (i, j) is shifted to a different position depending upon the computation. Instead of shifting the actual data around the storage array, we add/subtract co-ordinate values from the *rMap.x* and *rMap.y* values thereby shifting the data. Thus the new value of the pixel at (i, j) is given by (*rMap*(i, j).x, *rMap*(i, j).y). By the use of the *rMap* index we can compute the data transform independent of the data shifting process. At every instance during the computation, there is direct positional correlation between the initial image locations and the *rMap* index.

## 6 Equations for Data Shifting

In this section, we develop a generalized set of data movement equations, which can be applied to any block size. These set of equations constitute the complete DWT using the RMF algorithm. In Figure 1.1 we have a generalized block at any stage in the  $\log N$  steps taken to complete the process. We know from the discussion of the *Merge* operator the type of data block movement that is to be carried out. For example, consider block 1, shown in Figure 1.1.

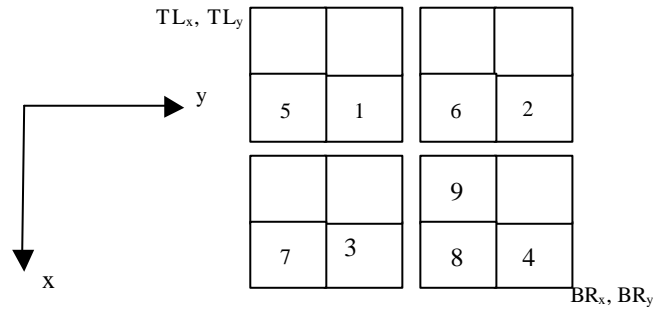


Figure 1.1 Block coordinate system and quadrants.

We know from the definition of the RMF algorithm that the data in block 1 has to be shifted to the position in block 9. Given any data item in block 1 in position (x, y), it is moved to position :

$$x + \frac{(BR_x - TL_x)}{4}, y + \frac{(BR_y - TL_y)}{4}$$

Where :  $(BR_x, BR_y)$  and  $(TL_x, TL_y)$  are the bottom right and top left coordinates of the block whose four quadrants are to be merged. Thus, we have managed to transform the data routing process into a simple arithmetic compute process. Similarly we define a set of primitives for the computation of the of the DWT using the RMF algorithm. These primitives define the operations to be carried out during the computation such as 1D-RMF, data shifting, and recursive operations.

## 7 Hardware-Software Architecture

The main aim of this architecture is to exploit the inherent parallelism between the data transforms and the data shifting process. This is done by a multi-threaded architecture using an FPGA. The overall architecture of the system is shown in Figure 1.2. We use two thread for process computations: one for the data compute and the other for the data routing. The data routing is carried out on the FPGA and the *rMap* index is stored on the RAM provided on the FPGA board. This allows faster access to the data for the data routing process carried out on the FPGA using an adder/subtractor circuit configured on the FPGA during the setup of the application. The main components of the architecture are :

- Primitive Block Computation Software Unit (PBCSU)
- Hardware-Software based Merge Process (MPS and the FPGA)
- Main memory based Queuing structure.

We briefly outline the working of the architecture. Initially the input image is stored in the main memory in an array of data values. The primitive block computation software unit (PBCSU) , reads 2x2 blocks from the main memory and computes the basic 2D RMF. However, the coordinates of the top left and bottom right of each 2x2 block computed by the PBCSU are written to queue Q1. This process continues as a thread until all the 2x2 blocks are computed and the thread then terminates. In addition to this thread, we also have the merge process software unit (MPSU) thread, which checks the status of queue Q1. Whenever the length of queue is  $> 4$ , the MPSU reads four sets of coordinates from the queue. These four coordinates represent the four blocks that are to be merged. The MPSU then carries out the data movement operation using the hardware configured on the FPGA. The data for the addition process is read from the FPGA on-board RAM. Once the merging process is completed, the coordinates of the merged block are written to queue Q2. The MPSU repeats the process for all 2x2 blocks read from Q1. The process continues until all the blocks in the queue Q1 are processed at which time all blocks of size 2x2 have been merged into 4x4 blocks. The MPSU then begins to read the block coordinates from the queue Q2, merges the blocks and writes the resulting coordinates to queue Q1. This process of switching between queue Q1 and queue Q2 is repeated until all the blocks are processed and we have a single entry in one of the queues.

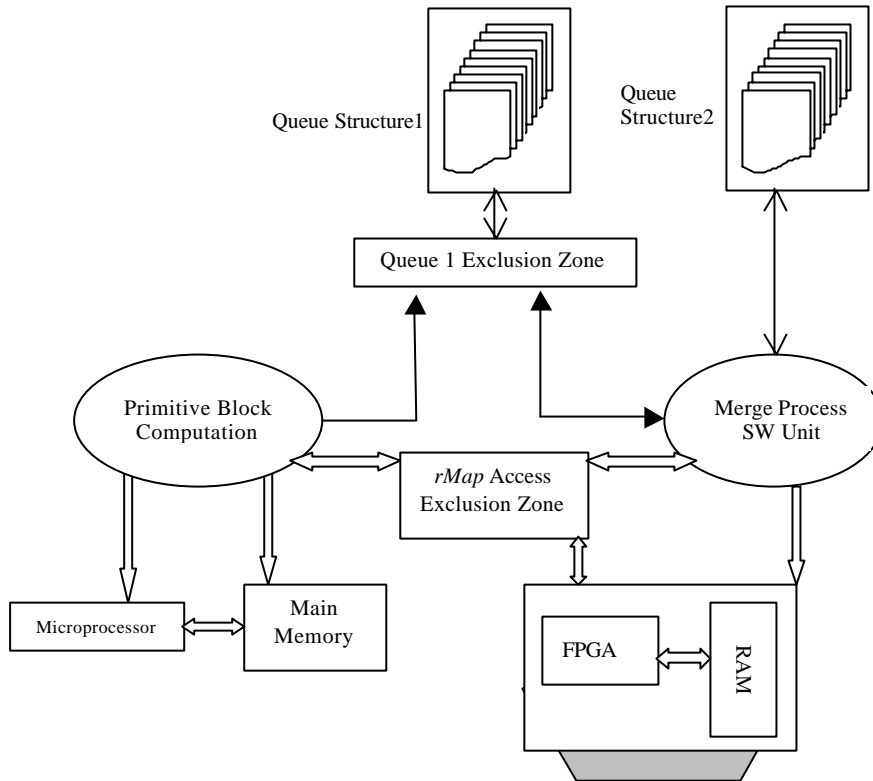


Figure 1.2 Hardware Software Architecture for DWT

## 8 FPGA Implementation and Resource Use

The H.O.T Works board from VCC[3] has been provided with onboard RAM, which can be used to store the *rMap* index. This technique of implementation of the *rMap* index use is efficient as the data shifting process can be carried out by the means of an addition/subtraction circuit configured on the FPGA. The *rMap* index contains a series of (x,y) pairs which point to a specific location in the original data matrix.

Figure 1.3 shows the comparative number of data accesses for the conventional RMF and the hardware-software implementation of RMF. We see a substantial decrease in the total number of direct accesses. Although we need to reset the data array to the correct positions after the completion of all blocks of a certain level, we can do so by using the block access mechanism rather than singular data accesses. This blocks access mechanism is a fraction of the initial data accesses. The figure below shows the original gray map image along with the reverse-transformed image (.PGM format).



Original Image <img-face.pgm>



Re-constructed Image

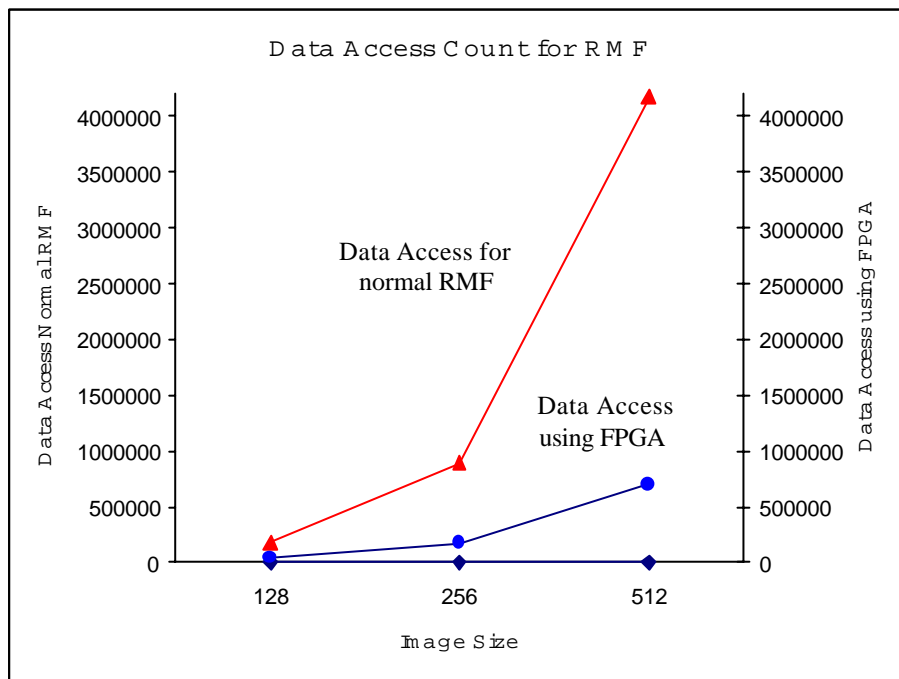


Fig. 1.3 Chart showing the reduction in the main memory data accesses. The data accesses are transformed into FPGA board RAM accesses.

### References

- [1] K. Mukherjee, "Image Compression and Transmission using Wavelets and Vector Quantization, Ph.D. Dissertation, University of Central Florida, 1999.
- [2] 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, July 1989
- [3] VCC H.O.T Works Board Manual